Aalto University – EURECOM
Master's Programme in Security and Cloud Computing

Establishing trust for secure elasticity in edge-cloud microservices

Établir la confiance pour une élasticité sécurisée dans les microservices en cloud de périphérie

Thesis submitted in partial fulfilment of the requirements for the degree of Master of Science by

Rohit Raj

Supervisors: Professor Hong-Linh Truong, *Aalto University*
Professor Aurèlien Francillon, *Eurecom*

Espoo, August 28, 2021

Aalto University
School of Science

EURECOM
Sophia Antipolis

Aalto University
School of Science

EURECOM
Sophia Antipolis

ABSTRACT

Author:	Rohit Raj		
Title: Establishing trust for secure elasticity in edge-cloud microservices			
Date:	August 28, 2021	**Pages:**	106
Major:	Security and Cloud Computing	**Code:**	SCI3113
Supervisors:	Professor Hong-Linh Truong Professor Aurèlien Francillon		

Platform services are increasingly becoming distributed to improve the availability and latency of Industrial Internet of Things (IIoT) applications. Modern infrastructure services such as Kubernetes have enabled a seamless deployment of these platform services across the distributed edge and cloud subsystems. These infrastructure services support dynamic addition and removal of resources, and thus, they enable the elasticity of the edge-cloud platform services. However, these infrastructure services currently do not have a high-level view of platform services and make elasticity decisions based on low-level configurations provided by the stakeholder.

This thesis aims to support trust establishment in the elasticity operations of these edge-cloud platform services. We present the ZETA framework that introduces Zero Trust Architecture (ZTA) secure design paradigm into these elasticity operations. ZETA ensures trusted elasticity of platform services via contextual Gaussian Process Regression (GPR) based trust computation from the "observed" and "service" knowledge. Moreover, it supports elasticity delegation capabilities through a token-based platform-agnostic interaction model. Finally, ZETA allows the stakeholder to provide custom trust policies, fine-tune the trust algorithm and even extend it.

The evaluation of the ZETA framework on multiple real-world scenarios demonstrates its ability to support zero-trust elasticity in variety of operations. Moreover, the encouraging results from the performance evaluation exhibit a low resource utilization and delineate precise resource requirements of ZETA provisioning.

Keywords: cloud computing, edge computing, distributed systems, zero-trust architecture, microservices, cloud security, microservice security, microservice elasticity

Language: English

Aalto University
School of Science

EUROCOM
Sophia Antipolis

RÉSUMÉ

Auteur:	Rohit Raj		
Titre: Établir la confiance pour une élasticité sécurisée dans les microservices en cloud de périphérie			
Date:	Août 28, 2021	**Pages:**	106
Programme d'Ètudes:	Security and Cloud Computing	**Code:**	SCI3113
Directeur de thèse:	Professeur Hong-Linh Truong Professeur Aurèlien Francillon		

Les services de plateforme sont de plus en plus distribués pour améliorer la disponibilité et la latence des applications de l'Internet Industriel des Objets (IIdO). Les services d'infrastructure modernes tels que Kubernetes ont permis un déploiement transparent de ces services de plateforme dans les sous-systèmes distribués de périphérie et du cloud. Ces services d'infrastructure prennent en charge l'ajout et le retrait dynamiques de ressources, et permettent ainsi l'élasticité des services de plateforme périphérie-cloud. Cependant, ces services d'infrastructure n'ont actuellement pas une vision de haut niveau des services de plateforme et prennent des décisions d'élasticité sur la base de configurations de bas niveau fournies par la partie prenante.

Cette thèse vise à soutenir l'établissement de la confiance dans les opérations d'élasticité de ces services de plateforme périphérie-cloud. Nous présentons le cadre ZETA qui introduit le paradigme de conception sécurisée Zero Trust Architecture (ZTA) dans ces opérations d'élasticité. ZETA garantit l'élasticité de confiance des services de la plateforme par le biais d'un calcul de confiance contextuel basé sur la régression du processus gaussien (GPR) à partir des connaissances "observées" et "du service". En outre, il prend également en charge les capacités de délégation de l'élasticité par le biais d'un modèle d'interaction agnostique de la plate-forme basé sur des jetons. Enfin, ZETA permet aux parties prenantes de fournir des politiques de confiance personnalisées, d'affiner l'algorithme de confiance et même de l'étendre.

L'évaluation du cadre ZETA sur de multiples scénarios du monde réel démontre sa capacité à prendre en charge l'élasticité sans confiance dans une variété d'opérations. De plus, les résultats encourageants de l'évaluation des performances montrent une faible utilisation des ressources et délimitent les besoins précis en ressources de l'approvisionnement ZETA.

Mots Clés:	cloud computing, edge computing, systèmes distribués, architecture à confiance zéro, microservices, sécurité du cloud, sécurité des microservices, élasticité des microservices
Langue:	Anglais

ज्ञानं परमं ध्येयम् ॥

- Knowledge is the supreme goal

Acknowledgements

I would first and foremost like to thank my supervisor, Professor **Hong-Linh Truong**, whose guidance was invaluable throughout the thesis. His insightful feedback pushed me to sharpen my thinking and brought my work to a higher level. He always pushed me to do the right thing, even when the road was tough. His expertise was vital in formulating methodology and identifying research questions. I would also like to express my appreciation towards my EURECOM supervisor, Professor **Aurèlien Francillon**. He was greatly supportive and always confident of my progress.

My education was supported by the SECCLO consortium of the European Union. I would like to thank them for providing this wonderful opportunity to participate in this double degree program. My experience in Finland has been extremely positive. For that, I am deeply grateful to the administrative staff: Prof. **Tuomas Aura**, **Eija Kujanpää**, **Anne Kiviharju** and **Laura Mursu**. Throughout these past two years, they meticulously ran the program and were always available for any help. I would also like to thank Dr. **Mohit Sethi** and Prof. **Mario Di Francesco**. They always had the best suggestions: both technical and on my future career.

I am also indebted to Prof. **Prabhat Kumar**, who, even after all these years has remained extremely positive of my abilities; perhaps even more so than me. My parents, family, and close friends have always stood beside me as a pillar of support. Their unconditional love, belief and support helped me navigate through the highs and lows of these last six months.

I am grateful for the cloud resources provided by CSC. These resources were vital in running the experimentations. Finally, I would like to thank all the researchers and open-source contributors who laid down the foundation of my thesis. Only by standing on the shoulders of of giants, I have seen further!

Finland, August 28, 2021
Rohit Raj

With the support of the
Erasmus+ Programme
of the European Union

Acronyms

ACE Arbitrary Code Execution. 53, 89, 90
ACID Atomicity Consistency Isolation and Durability. 71
API Application Programming Interface. 38, 39, 53, 73, 75
CNN Convolutional Neural Network. 26
CSP Cloud Service Provider. 14, 25, 31, 35, 37, 41, 52, 83
DNN Deep Neural Network. 38, 89
DSLAM Digital Subscriber Line Access Multiplexer. 29
EOP Escalation Of Privilege. 53
GPON Gigabit Passive Optic Networks. 15, 20, 29, 30, 31, 32, 39, 41, 80, 81, 88
GPR Gaussian Process Regression. 11, 18, 22, 23, 27, 58, 59, 60, 62, 66, 75, 88, 89
HPA Horizontal Pod Autoscaler. 43, 77, 78, 81
IIoT Industrial Internet of Things. 13, 15, 25, 29, 32
IoT Internet of Things. 13, 29, 31
JWT JSON Web Tokens. 54, 63, 65, 68, 71, 73, 93
ML Machine Learning. 21, 25, 26, 38, 39, 41, 42, 75, 90
MQTT Message Queue Telemetry Transport. 15
OLT Optical Line Terminal. 29
ONT Optical Network Terminal. 29
PON Passive Optic Networks. 29
RBF Radial Basis Functions. 23, 60
REST REpresentational State Transfer. 38, 39, 53, 55, 70, 71
SLA Service Level Agreements. 15, 19, 25, 26, 32, 38, 53, 83
SLI Service Level Indicator. 53
STIX Structured Threat Information eXpression. 58
TEE Trusted Execution Environments. 25, 90
TLS Transport Layer Security. 54
VM Virtual Machine. 14, 20, 24, 30, 32, 75, 91, 92, 94
VNF Virtual Network Functions. 20, 75
ZETA ZEro Trust elAsticity. 17, 52, 55, 58, 71, 72, 73, 74, 75, 77, 87, 89, 90, 91, 92
ZTA Zero Trust Architecture. 14, 21, 26, 28, 32, 38, 42, 51, 53, 67, 68, 92, 93

Contents

List of Tables

List of Figures

Chapter 1

Introduction

"*Technology trust is a good thing, but control is a better one*"
– Stephane Nappo

1.1 Context

To ensure high availability and reliability of platform services with latency guarantees, a modern Industrial Internet of Things (IIoT) platform service infrastructure is typically decomposed into three major subsystems. They include the Internet of Things (IoT), edge and cloud subsystems. Each of these subsystems is characterized by varied computing capabilities. Cloud is often located in a remote data-centre and boasts of high-scalability and theoretically unlimited storage capabilities [79]. On the other hand, edge subsystems are more recent extension to cloud and are usually situated in proximity to the IoT systems (such as base station, routers) [72]. They are especially useful for providing latency-sensitive computational offloading services [72, 87] with limited scalability and storage capabilities.

To service the IoT subsystem, multiple platform services run atop the edge-cloud infrastructure. These platform services often constitute of multiple distributed microservices [69]. Microservices are disintegrated but highly cohesive services that typically perform a single operation. Moreover, containerization of these microservices provides an abstraction for uniform deployment across diverse kinds of edge-cloud systems while maintaining low resource overhead [86]. Examples of platform service include message brokers (such as Apache Kafka[1]), stream processors (such as Apache Flink[2]), etc.

As the organizations diversify their deployment, dedicated infrastructure services are being extensively used by them for proper management of their plat-

[1] https://kafka.apache.org
[2] https://flink.apache.org

form services across heterogeneous subsystems [83]. These infrastructure services share common control plane for resource allocation and management across subsystems [40]. Often these infrastructure services are present as container orchestration framework such as Kubernetes[3]. Other techniques involve resource management of Virtual Machine (VM) [33]. Nonetheless, the ability to add or remove resources on *ad-hoc* basis is one of the cornerstone of edge-cloud infrastructure services and forms elasticity of the edge-cloud continuum. However, elasticity is a multi-dimensional concept [17]. Any effect on resource elasticity (due to security or otherwise) may have effect on resilience and availability of these platform services.

Modern infrastructure services support *affinity-aware offloading* [66] and enable dynamic elasticity of resources while maintaining intra-operability. However, this dynamic elasticity is also associated with major research challenges. As per [32], one of the major research challenge is ensuring trust in the elasticity operations of these infrastructure services. There is a complex dependency between the microservices of an edge-cloud system [43] and any security attack (elastic or otherwise) has a potential to cascade across the edge-cloud system. For example, if a message broker on edge is compromised, the attacker can alter the rate of data publication. A downstream microservice that consumes the data from the aforementioned compromised message broker may elastically scale itself to adjust to this new rate of message publication. From [15], we know that situations such as these pose security threats such as lateral microservice movement or data exfiltration. Moreover, this situation can also cause resource and monetary wastage to the organizations and Cloud Service Provider (CSP). Therefore, to avoid such situations, it is imperative to have a high-degree of trust on any elasticity operation.

This thesis aims to support trust establishment in the elasticity operations of the edge-cloud platform services. Infrastructure services currently lack the high-level view of elasticity and make elasticity decisions based on low level configurations provided by the stakeholder. Ideally, any elasticity framework should have a high-level view of all the platform services to dictate elasticity logic to infrastructure service. In essence, we must take away the elasticity logic from the infrastructure service and pass it a system that does have such a high-level view. Such a system is said to follow Zero Trust Architecture (ZTA) security design paradigm which is based on "trust no-one" approach and limits internal lateral movement [67]. Therefore, to avoid such situations, an edge-cloud service can utilize ZTA paradigm for its elastic operations.

Our work aims to provide support to the primary stakeholders (which are edge-cloud platform service developers) by improving the trust of elasticity of edge-cloud microservices through a framework that introduces ZTA security paradigm into the elasticity operations of the edge-cloud infrastructure service. Other secondary

[3] https://kubernetes.io

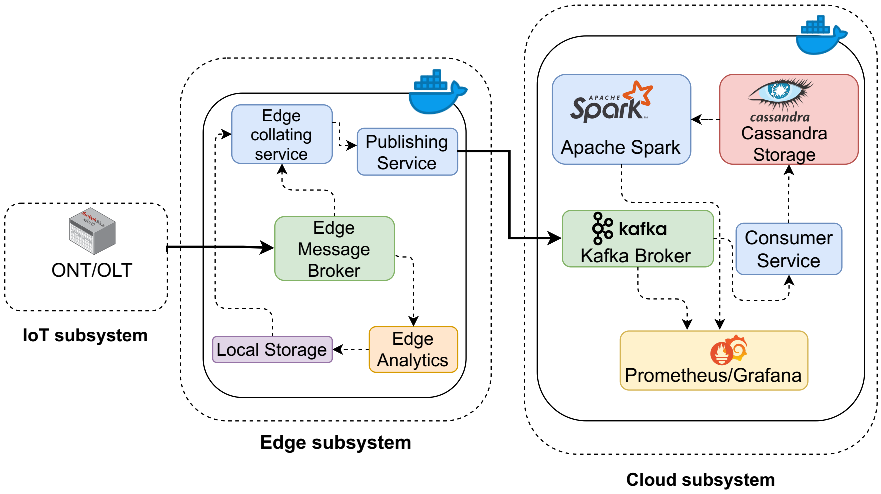


Figure 1.1: Subsystems and platform services in GPON edge-cloud monitoring infrastructure

stakeholder may include the cloud services provider since they are typically bound by Service Level Agreements (SLA) and hence need to ensure certain level of elasticity and robustness in their resource services. Unlike traditional ZTA use-cases such as [53, 67], we cannot rely on manual multi-factor authentication due to stricter automation requirements in elasticity of distributed systems.

1.2 Motivating use-case

Here we present a motivating use-case of GPON to further illustrate the need of high-trust in elasticity operations of the infrastructure services. Let us consider a system that monitors and analyzes the status of equipments in an ISP network. We can see from Figure 1.1 that this monitoring infrastructure is structured as a typical IIoT infrastructure with three different subsystems. The GPON ISPs infrastructure consist of many equipments. Later in this thesis (Section 3.2), we discuss the GPON monitoring infrastructure and the data (type, amount, and flow) in more detail.

The edge and cloud subsystem of this monitoring infrastructure provides reliable analytics to the GPON developers. To support these analytics, the GPON developers use different platform services. For example, the edge subsystem consists of a MQTT broker that captures information from the sensor subsystem. The local edge analytics is a containerized microservice that publishes the results to the cloud subsystem. The cloud broker (Apache Kafka) has the ability to scale up and down depending on the rate of data publish requests coming from the edge subsystem's microservice.

While the security of these services are usually carefully considered, "security" of elasticity operations is often overlooked. As with most cases, the infrastructure services use simple threshold of resource usage as elasticity logic. In the edge-cloud subsystems shown in Figure 1.1, it is usually 60–70% of CPU or memory resource usage, before extra resources are allocated to compensate the increase in demand. In such situations, if an attacker is able to gain access to edge subsystem, they can cause unnecessary scale-up or scale-down operation on the cloud broker simply by varying the rate of data publication. This is a very simple but classic mapping of how a security incident on edge can indirectly affect the elasticity of cloud broker. Moreover, due to the dynamic nature of such edge-cloud systems, it is not practical for the developers to manually monitor or work on the elasticity operations.

From the example above, we identify three basic problems for stakeholders. That a) the trust in elasticity operations is often implicit and checked at a low-level by the infrastructure and that there is a no proper holistic and high-level view of trust. Another issue is b) the lack of a proper framework architecture that can evaluate trust on a high-level and multidimensional observed data metrics and c) the lack of trust-level customization and granular policy enforcement for stakeholders' specific use-case.

1.3 Research questions and requirements

Through this thesis, we aim to improve trust in elasticity operations of a service. Towards that end, we aim to answer the following research questions:

- RQ1: **Architecting the trust in elasticity**: What should be the guiding principle of trust in elasticity operations? More importantly, is there any framework that has a high-level view of the platform services (and infrastructure services) and supports elasticity operations for infrastructure services. Finally, how can this high-level "trust" be calculated.

- RQ2: **Ensuring trust in interaction**: How can we ensure that the interaction between services and framework is non-compromised. How can we model and represent the interaction between for diverse resources in edge-cloud systems with high-integrity.

- RQ3: **Customizable policies, fine-tunable trust and extensible trust evaluation**: How can the stakeholder create custom policies and enforce them with microservice-level granularity. Moreover, how can we support stakeholders to fine-tune the trust levels via configuration files and utilize the output as policy. Finally, how can we allow stakeholders to extend the trust algorithms and supplant with their own?

1.4 Approach

To reliably solve the problem of trusted elasticity and answer the previously mentioned research questions, we adhere to the following approach:

- We present real-world motivating scenarios and collate use-cases based on them. These use cases serve as the basis of functional and non-functional use-case requirements of our framework.
- Based on the functional requirements, we design a framework named ZETA to improve trust of elastic operations of edge-cloud platform services.
- We describe an interaction model that allows the infrastructure services to utilize ZETA framework securely and with integrity. This model is platform-agnostic.
- We present techniques to calculate trust level and evaluate policies with a high-level view of the deployed platform services.
- We present methodologies to allow the stakeholder to manage their custom trust policies and fine-tune their trust levels. We also present techniques to extend ZETA framework by executing external trust algorithms of stakeholder's choice.
- We implement the ZETA framework and evaluate its feasibility on multiple real-world scenarios. We also perform performance evaluation of ZETA framework and discuss these evaluation results.

1.5 Contributions

Through this thesis, we aim to provide answers to the research questions that we raised in Section 1.3. Accordingly, we state the following contributions of this thesis:

- **ZEro Trust elAsticity (ZETA) Framework**: We contribute a comprehensive ZETA framework which has following features:
  - ⧫ Ensure support of **trusted elasticity** in distributed edge and cloud platform services with a token-based interaction model
  - ⧫ **Platform-agnostic** interaction model support for infrastructure services with both shared control plane (homogenous) and data plane (heterogeneous)

- **Elasticity delegation** capabilities through the use of a combination of two distinct JWT tokens
- **Contextual** and multi-dimensional uncertainty and trust-level computation through captured **observed and service knowledge**
- Default **trust computation** algorithm that uses Gaussian Process Regression (GPR). Our algorithm supports near real-time trust level sampling
- **Extensibility** of trust computation by allowing providers to supplant their own custom algorithm with the default GPR trust algorithm
- **Customizability** in trust level management through simple configuration files
- Microservice level **granularity** in trust-enforceability through support for user-defined policies

We implement the prototype of the ZETA framework and open-source it into the GitHub repository at https://github.com/rdsea/ZETA. This repository contains all the code, scripts, testing utilities and the documentation of ZETA framework. Additionally, we perform the evaluation of framework on two distinct real-world scenarios. This functional evaluation is aimed at proving the usefulness of the framework. We also validate the performance, latency and quantify resource utilization through stress test of ZETA framework. Together with scenario based evaluation and performance evaluation, we address the two most important aspects (functional and non-functional) of software testing [71].

This design and capabilities of this framework is based on our previous work on the understanding of security-elasticity dependency analytics in edge-cloud microservices [61]. The knowledge gained and lessons learned during the development and evaluation of SEDAICO framework[4] were pivotal in the development of ZETA framework.

1.6 Thesis structure

Chapter 2 gives the background knowledge and the related work in the domain of trusted elasticity. Chapter 3 introduces the motivating scenarios and the use-cases. We extract the functional and non-functional requirements of our framework in this chapter. The models, designs, workflows and the implementation of ZETA are presented in Chapter 4. Chapter 5 presents the evaluation results and discusses the security and configurability of the framework. It also examines the lessons learned during the evaluation. Finally, the thesis presents the concluding remarks and explores the future work in Chapter 6.

[4] https://github.com/rdsea/SEDAICO

Chapter 2

Background and related work

2.1 Overview

In this chapter, we present the background focusing on trust in the elasticity in edge and cloud services. This will help us establish a research gap and delineate state-of-the-art methodologies that are being used to ensure trusted elasticity in these edge-cloud services.

Since this thesis focuses on enhancing trust in elasticity operations, our main review of related work pivots around it. Moreover, we also present literature review on resource elasticity and methodologies to promote trust (not necessarily only elastic trust) between subsystems and microservices in an edge-cloud system; as they provide a pivot for our work to build upon.

2.2 Background

2.2.1 Elasticity

One of the major guarantees that most cloud providers give in their SLA is "elasticity" of resources [17, 51, 61, 68]. Elasticity in the edge and cloud computing scenario is often misnomered as simple addition/removal of resources on ad-hoc basis. However, elasticity is a multi-dimensional aspect and can be applied at many levels in our edge-cloud continuum [21, 68, 78].

2.2.1.1 Multi-Dimensional elasticity

The elasticity in edge-cloud continuum can be broadly classified as *resource elasticity*, *cost elasticity*, and *quality elasticity* [21]. Resource elasticity is the classic notion of elasticity which allows dynamic resource provisioning into the infrastructure. These resources could be CPU, memory, VMs, VNFs, etc. Cost elasticity on the other hand is the variance of cost with reference to the responsiveness of the resources. In this elasticity, the resource allocation under the constraint of cost is measured. For example, how will the resources be distributed under a total budget of *X euros*. Lastly, quality elasticity deals with measurement of quality of service with the change in resource usage. This dimension of elasticity focuses on the trade-off between resource deployment and quality of services [78]. Our thesis, however, solely focuses on resources elasticity.

2.2.1.2 Multi-level elasticity

The elasticity can also be categorized at the different levels of granularity. These categorizations include cloud service, service units, code regions [17]. At *cloud service* level, the elasticity of the whole application such as complete GPON monitoring infrastructure is measured. At the *service unit* level, the elasticity of individual service such as a database is the focus. On *code unit* level, the elasticity at individual units of executable code is measured. This thesis focuses on service elasticity.

2.2.2 Trust in systems

Trust is a very subjective term and very difficult to guarantee[1]. As per Dieogo Gambetta's definition of trust in [23], *"Trust is a subjective probability with which an agent will perform a particular action ..."*. It is important to note here that trusts are not necessarily transitive. That is, if Alice $\xrightarrow{\text{trusts}}$ Bob and Bob $\xrightarrow{\text{trusts}}$ Carol then Alice $\xrightarrow{\text{trusts}}$ Carol may not hold true. In terms of distributed system, the notation of trust is even more elusive. The notion of trust in a distributed system is popularly accepted as a presence of proper access rights between nodes [9].

2.2.2.1 Trust in the context of distributed systems

According to [85], trust between nodes of a distributed system implies that any hosts expects the operation requests from a "trusted" source are benevolent with a high-degree of confidence. This confidence can be improved by improving the security of the operation requests.

Distributed trust in edge-cloud services (systems) can modelled as *implicit* and *explicit* (See Figure 2.1). In *implicit* trust, the service unit determines the trust of any request. Verification of cryptographic certificates of any entity or request is

[1] *"There's no test possible that can prove the absence of flaws"*: Bruce Schneier. *From*: Why Cryptography Is Harder Than It Looks.

an example of implicit requests. In case of *explicit* trust, a level of indirection in form of a trusted third-party with security policies is added. These policies can be defined through various external sources such as ML models, stakeholders, human experts, etc.

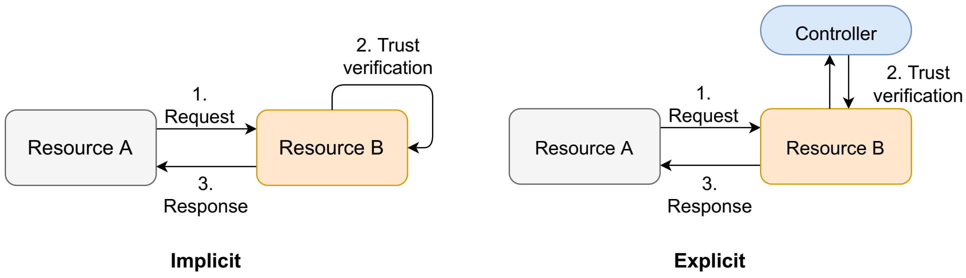


Figure 2.1: Implicit vs explicit trust in distributed environment

2.2.3 Platform services vs infrastructure services

One of the distinctions that we make in this thesis is that of platform vs infrastructure services. Platform services execute the developer application whereas infrastructure services provide resources to run these platform services. The resource elasticity of platform services are managed by these infrastructure services. As the elasticity is controlled by the infrastructure services, the framework must always interact with infrastructure services and never with platform services. However, it is always the elasticity of platform services that is evaluated.

2.2.4 Zero trust architecture

Zero Trust Architecture (ZTA) (Formalized in *NIST's 800-207* special publication) is defined in [67] as a cybersecurity paradigm that is designed around "trust no one" principle. In this security design paradigm, trust is never granted implicitly but must be continuously assessed. As per [11] and [67], any ZTA based architecture *MUST* have following three components:

1. *Policy Engine (PE)*: grants the decision to a subject for access to any object
2. *Policy Administrator (PA)*: Acts on the decision of PE. Generates credentials or additional instances in case of access grants.
3. *Policy enforcement point (PEP)*: The endpoint for a subject to request decision from PA.

Use of ZTA paradigm forms the backbone logic of our framework. Figure 2.2 presents a visual description of the elasticity of a system that follows ZTA paradigm.

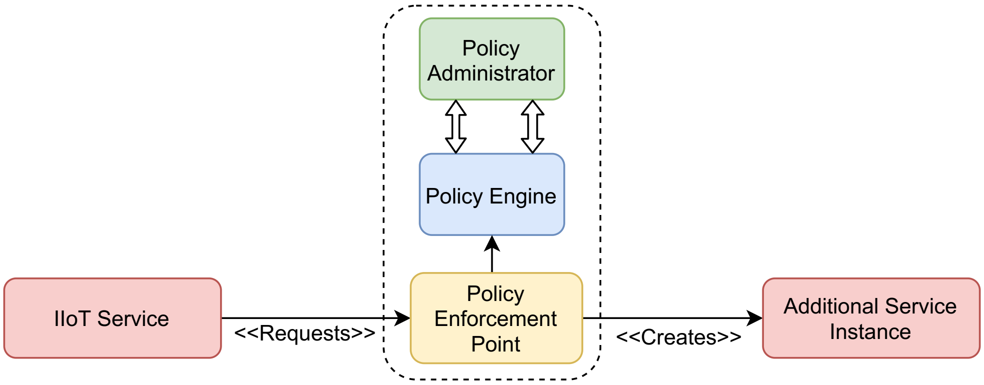


Figure 2.2: ZTA design paradigm [11]

2.2.5 Gaussian processes for trust

We use gaussian processes for calculating contextual trust levels from the observed data. A gaussian process generalizes the "*Gaussian probability distribution*" [63]. Unlike regular non-linear regression methods, a GPR is also able to provide its own uncertainties. Another distinctive property of bayesian (and gaussian) distributions are that they are closed under conditioning and marginalization [26]. Hence, any output derived from the GPR would follow a similar probability distribution. Using these distributions with mapping of uncertainties to confidence intervals is an intuitive way to define boundaries of trust.

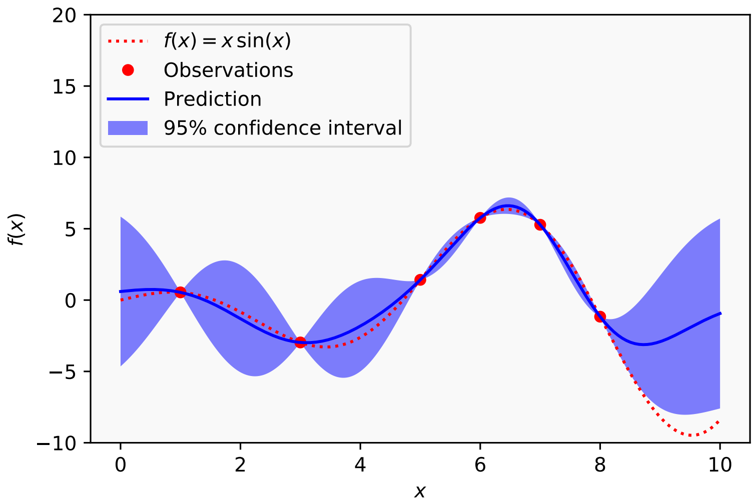


Figure 2.3: A sample GPR curve for the function $xsin(x)^*$
*(Representative image generated from the sample code of [56])

Figure 2.3 presents a sample regression of the function $f(x) = xsinx(x)$ using GPR. A GPR is represented by the Equation 2.1 and requires calculation of a symmetric covariance matrix (S) [63] and mean (m). Here, **f** is the function value, X_* are the test inputs, and X are the training (observed) points. Finally, $K_{(X_*,X)}$ is a matrix of shape ($X_* \times X$). Further details on gaussian processes, kernels and the derivation of Equations 2.1, 2.2, and 2.3 can be found in Carl Rasmussen & Christopher William's book at [63].

$$\begin{aligned}
&\mathbf{f} \mid X, \mathbf{y}, X_* \sim N(\mu, S), \text{ where} \\
&\qquad \mu = K_{(X_*,X)}[K_{(X,X)} + \sigma_n^2 I]^{-1}\mathbf{y} \\
&\qquad S = cov(\mathbf{f}) = K_{(X_*,X_*)} - K_{(X_*,X)}[K_{(X,X)} + \sigma^2 I]^{-1} K^T_{(X_*,X)}
\end{aligned} \tag{2.1}$$

2.2.5.1 RBF kernels

The covariance matrix from the previous equation is also called the kernel. GPR "learns" the non-linear function in the linear space defined by the kernel. Kernels are also used in other domains of machine-learning such as Support Vector Machines [4]. RBF[2] kernels are one such stationary kernels that use an exponentiation of the scaled squared Euclidean distance.

$$\begin{aligned}
K_{(X_*,X)} &= \exp\left(-\frac{||X_* - X||^2}{2\sigma^2}\right) \\
&= \exp\left(-\gamma ||X_* - X||^2\right)
\end{aligned} \tag{2.2}$$

Equation 2.2 presents the RBF kernel, where γ is equal to $1/2\sigma^2$. Our thesis uses RBF kernel for the GPR as it can tuned easily. Moreover, as it is a *stationary* kernel and therefore its output remains same even after subjecting it to translation.

2.2.5.2 White noise kernel

White noise kernel is an ancillary kernel that represents the noise of the training data as independent and identical. This work utilizes the Equation 2.3 from *WhiteKernel* of scikit-learn [56] to represent the white noise. Here N_l is the noise level.

$$K(X_1, X_2) = \begin{cases} N_l, & \text{if } x_i = x_j \\ 0, & \text{otherwise} \end{cases} \tag{2.3}$$

[2]RBF kernels are also called *squared exponential*

2.3 Related work

2.3.1 Trust models for edge-cloud services

While there is a plethora of literature on edge-cloud security, limited works attempt to model trust in an elastic cloud infrastructure. The work by Y. Jarraya et at. [30] provides systematic mechanism to model and verify the security of VM layer in cloud. More specifically, they focus on modelling the VM layer deployment via an algebraic calculus framework. Sahli et al. [68] model the cloud elasticity through non-conventional bi-graphical reactive systems. They divide the cloud services into two zones namely frontend and backend. It then creates the elasticity models through a bigraph that is represented as a set of reaction rules between these two. This approach, however, lacks the ability to represent elasticity in a logical, graphical and rigorous way. CPL [10] is a statically typed cloud programming language that focuses on fault-tolerance but also takes into consideration the elasticity models of cloud and edge deployment. Chunlin Li et al. [38] propose a limited model of distributed edge-cloud resource load prediction and management for both vertical and horizontal elasticity.

RECAP [54] models reliable capacity provisioning of edge and cloud services. It includes a "RECAP application modeller" that defines the edge cloud services and their respective QoS requirements. It additionally includes components like workload modeller, optimizer and scheduler to execute those resource workloads efficiently. Modelling of elasticity in edge-cloud systems is not limited to the perspective of trust or security. There are attempts to model elasticity based on energy efficiency. Works such as [8, 29, 51] model the elasticity of cloud services based on energy efficiency of the platform. [8] and [51] models the elasticity scaling with energy efficiency for SaaS, PaaS and IaaS platforms whereas [29] models only for PaaS and IaaS services.

While we draw inspiration from different approaches taken to model trust in elasticity in cloud computing paradigm, unfortunately most of these work limit their scope to either cloud or edge subsystems and do not propose a model that unifies both edge and cloud. Our work on the other hand is both subsystem and platform agnostic. That is, unlike works such as [8, 10, 29, 51, 54], our work can support all platform service models. Furthermore, unlike [30, 68], ZETA can support distributed edge and cloud platform services.

2.3.2 Trust in elasticity of edge-cloud system

There are relatively few literatures that take into consideration the trust parameters in determining service deployment on the edge and cloud nodes. One of the first approaches to address trust in cloud deployment was the work by J. Luna et al. in [41] and [42]. They propose *Quantitative Policy Trees* [42] that allow mapping and processing cloud security level agreements through a data structure. They further expand their work in [41] to add *Quantitative Hierarchical Processes* and rank cloud service providers by security levels provided in their SLA. `TRUST-CAP` [2] attempts to evaluate behaviours and recommend cloud resources to the user. `TRUST-CAP` has separate components to provide trust in its elasticity recommendation (such as privacy, integrity) based on collected metrics. However, this model is only evaluated with man-in-the-middle and man-at-the-end attacks. Our work on the other hand, can also mitigate lateral movement and data exfiltration and denial of services attack.

S. Forti et al. propose `SegFog` [22] which is a framework to improve trust in secure deployment of edge-cloud services. `SegFog` uses `ProbLog`[3] library to define probabilistic trust models. To use `SegFog`, any stakeholder (CSP and application operator) needs to provide extra *Node Descriptor* files that contain information about the "Security Capabilities" and "Security Requirements". Additionally, they also need to provide the composite trust values of each node that they have or require. Since this trust framework requires declaration of a pre-defined trust values from the CSP, multiple malicious CSP's can therefore collaborate to attack the trust model. The ZETA framework evaluates every trust value in isolation from the CSP and hence is hardened against such attacks.

Other literatures attempt to address elasticity of trust only on a specific edge-cloud resource types and fail to provide a uniform framework that can operate on rich use-cases and resources across the edge-cloud continuum. For example, `Diggi` [25] framework allows deployment of secure serverless functions through the use of TEEs (Intel's Software Guard Extensions [18] in this case). However, this requires all the machines to be build upon SGX. Our work, however, does not have any such hardware requirements. B. Qolomany [58] propose a trust based machine learning model selection and deployment for IIoT and smart city services. This work, however, only focuses on initial deployment of models and not about the trust in subsequent elasticity which is addressed by our work. `ENORM` [82] is another framework that provides resource management feature to the edge-cloud services. This framework focus on reducing the latency of the resource provisioning and scaling on the edge systems with online games as the use-case. However, unlike the token-based approach of ZETA, such a system does not take security aspect into consideration

Finally, there has been a rise in use of artificial intelligence and Machine Learning (ML) for establishing trust. PREPARE [76] uses a tree-augmented naive bayesian network for resource metric value prediction. More recent works leverage the power of deep-learning for these predictions. The work in [24] proposes a

[3]https://github.com/ML-KULeuven/problog

hybrid model using CNN and grey wolf optimization. It uses a relatively simpler 4 layers of CNN networks and provides an F-score of more than 99%.The work in [84] performs a resource offloading based on n neural networks and training each of them with a randomly batched data from the database. However, these approaches revolves around anomaly detection (such as [24, 50, 60, 76] etc.) or resource optimization (such as [84]) through captured data and is often passive i.e. it needs to train a model. However, all these AI/ML models treat outliers as anomalies and anomaly detection is an indirect way of establishing trust. Other important aspect is that these frameworks fix the ML model used for establishing trust and does not allow the user to fine-tune trust level or use custom ML models. ZETA addresses both these shortcomings, i.e. direct context-based trust evaluation and complete trust configurability to the stakeholder. While ACAS [48] framework does allow fine-tuning, it does not provide extensibility feature and has limited support for policies. Commitment [3] framework uses a novel trust evaluation algorithm. However, it does not provide support for custom policies and trust customizability. Konstantinos et al. [55] propose a SLA and reputation based trust model and a framework based on this model for provisioning of resources on cloud. It relies on a long-term reputation of a resource from a providers based on their key performance indicators. However, similar to `SegFog` [22], such a approach will be vulnerable to long-term collaboration attacks. As our work computes the trust level both on short and long term observed and service metrics, it is relatively hardened against such attacks.

2.3.3 Elasticity in edge-cloud infrastructure services

The current auto-scaling solutions for the infrastructure are too low-level [14]. That is, they use a configuration-first approach. Developers embed the policies and rules into the deployment configurations and the infrastructure's control plane drives the elasticity based on them. Few works such as [6, 35] provide support for external policy evaluation for the auto-scalers. While this approach allows an easier management of scaling operations, we argue that providing ZTA support on elasticity is difficult due to static nature of configuration files. Firstly, this approach does not decouples the auto-scaling logic from the deployment, meaning that third-party trust cannot be established. Secondly, none of the approaches perform a multivariate trust analysis from the observed data and finally, this approach does not work well with heterogeneous solutions. That is, due to a lack of a centralized and third-party trust establishment framework, elasticity requests of infrastructure services cannot be interchangeably requested by different infrastructure services. For example, a Kubernetes horizontal pod auto-scaler cannot and should not be trusted to scale on a request from docker-swarm auto-scaler due to violation secure design principles of Saltzer and Schroeder [73].

2.4 Summary

This chapter provides the background necessary for this thesis. As our framework attempts to build trust for the elasticity operation in distributed microservice, we also formally introduce the notion of trust for distributed systems in this chapter. We present the methods of GPR and provide an idea why they are suitable for calculation of trust levels in our framework. This chapter also briefly introduces the concept of how the gaussian processes operate and the function of the kernels involved in them. We succinctly introduced two kernels that will later be used during the implementation of GPR in the framework.

To the best of our knowledge, there is no descriptive state-of-the-art research work that aims to provide answer to the research questions raised in Section 1.3. That is, none of these researches :

- Model *interaction techniques* that provide trusted elasticity operations across distinct scenarios and use-cases in an edge-cloud services. Most of them enforce trust in a domain-specific subset of use-cases.

- Provide a unified framework for managing the trust of elastic operations along with providing the capability of customize it to different stakeholders.

Chapter 3

Trusted elasticity requirement analysis

3.1 Overview

Our aim in this thesis is to support stakeholders in increasing the trust of elasticity operations in their platform services. Towards that end, we consider two real-world scenarios and their problems to derive requirements of our proposed framework. Through these use-cases, we gather, extract and analyse the trusted elasticity requirements that our framework must support. They cover a variety of aspects on the architecture capabilities and limitations. Using these different deployment models allows us to accommodate varied use-cases and thereby evaluate our framework across wide spectrum of trusted-elasticity problems.

This chapter presents two real-world edge-cloud infrastructure scenarios. They are diverse in their operations, platform services and the types of deployment infrastructure service used. In each of these scenarios, we define multiple use-cases along with the stakeholder, preconditions, activities, alternative activities and end-goals. Based on these use-cases and the research questions discussed in Chapter 1, we then derive concrete requirements of our framework.

3.1.1 Elasticity in the scenarios

To improve the trust in elasticity operations, our framework incorporates the design principles of ZTA and uses explicit trust. All the infrastructure services of an edge-cloud subsystem must require approval from the framework before any elasticity operation. Figure 3.1 presents a high-level view of a service's elasticity operations. This is applicable for all the services in both the scenarios. We will delineate the implementation details and request-response of the framework in subsequent chapters.

To ensure trust in the elasticity operations in these scenarios, the requests must be approved only through pre-determined rules by domain experts. Distribution of subsystems adds a level of indirection and exposes the attack surface area. Therefore, the rule maintenance must be centralized to ensure higher security of the component. We can say that as long as this centralized PEP is secure, we can trust the elasticity operations. Moreover, making it centralized also ensure that the rules are easier to manage and maintain. Finally, having a centralized PEP in also aligns well with the secure design principles of Saltzer & Schroeder [73].

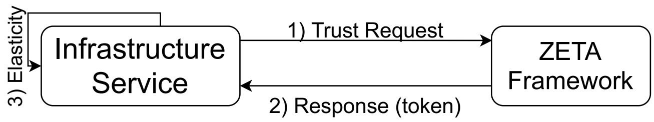


Figure 3.1: Interaction with ZETA framework

3.2 Scenario: GPON network monitoring

Our first scenario is of the equipment monitoring subsystem. This scenario has been developed by us and is based on our work on security-elasticity analytics in [61]. GPONs are point to multi-cast networks and are extensively used to provide the fibre to home services by the ISPs. GPON uses passive splitters inside the network and multi-casts the downstream network data to all the terminal endpoints called ONT. These GPON networks infrastructure comprise of multiple equipments such as PON, DSLAM. In order to provide internet services, the data from these devices goes via PON ports to an OLT terminal.

A PON port can handle 64 ONU whereas the OLT can have 2048 ONU. Hence, each OLT can be used for around 700–800 customers. The OLT publishes the data at the rate of 1 data per sensor every 5 minute. Appendix A summarizes the data schema and sample data of the IoT subsystem published onto the edge broker in the Listing A.1 and Table A.1 respectively. Note that the GPON infrastructure comprise our IoT subsystem. The GPON monitoring consist of our edge and cloud subsystem.

As discussed in the introduction, a modern IIoT infrastructure consists of multiple subsystems. The GPON monitoring system uses edge and cloud subsystems with **“n→1”** flow. That is, multiple IoT subsystems publish to the edge, whereas multiple edge systems connect to a single cloud. This means that the edge devices do not interact within themselves and that there is a single tier of edge subsystem. This is in contrast to the other scenario that we discuss later.

Subsystem	Platform services Units	Infrastructure service units
Edge	Message Broker, Data publishing and consuming microservices	Containers
Edge & Cloud	Stream Processing service, data collation service	Containers and VM
Cloud	NoSQL Columnar database	VM and Containers
Cloud	Dashboard and monitoring service	Containers

Table 3.1: Services and elastic infrastructure in the GPON scenario

3.2.1 Platform and infrastructure services

Each subsystem consists of multiple platform services that interact among themselves through internal/external network layer. These platform services are deployed on an infrastructure service with elastic capabilities for proper management of resources. The various microservices along with the elastic infrastructure services involved in these subsystems are summarized in Table 3.1.

This GPON monitoring provisioning can be found in the `IoTCloudSamples`[1] repository [77].

3.2.1.1 Elasticity of services

The platform services are scaled through elastic infrastructure service units. We present the scope of elasticity of the two different elastic infrastructure service units (containers and VM) indicated in Table 3.1. Containers can be scaled vertically and horizontally by allocating more resources such as memory, CPU, etc. These containerized platform services can be scaled horizontally by creating additional containers. For example, we can add extra nodes to the stream processing services by creating extra container and connecting the worker nodes with the master node. VMs on the other hand support vertical scaling only.

3.2.2 Stakeholders

We have identified the following three stakeholders in the GPON monitoring system:

- **Developers**: They are responsible for creating and maintaining the platform services across the edge and cloud subsystem. Additionally, they are also

[1] https://github.com/rdsea/IoTCloudSamples/tree/master/scenarios/netops

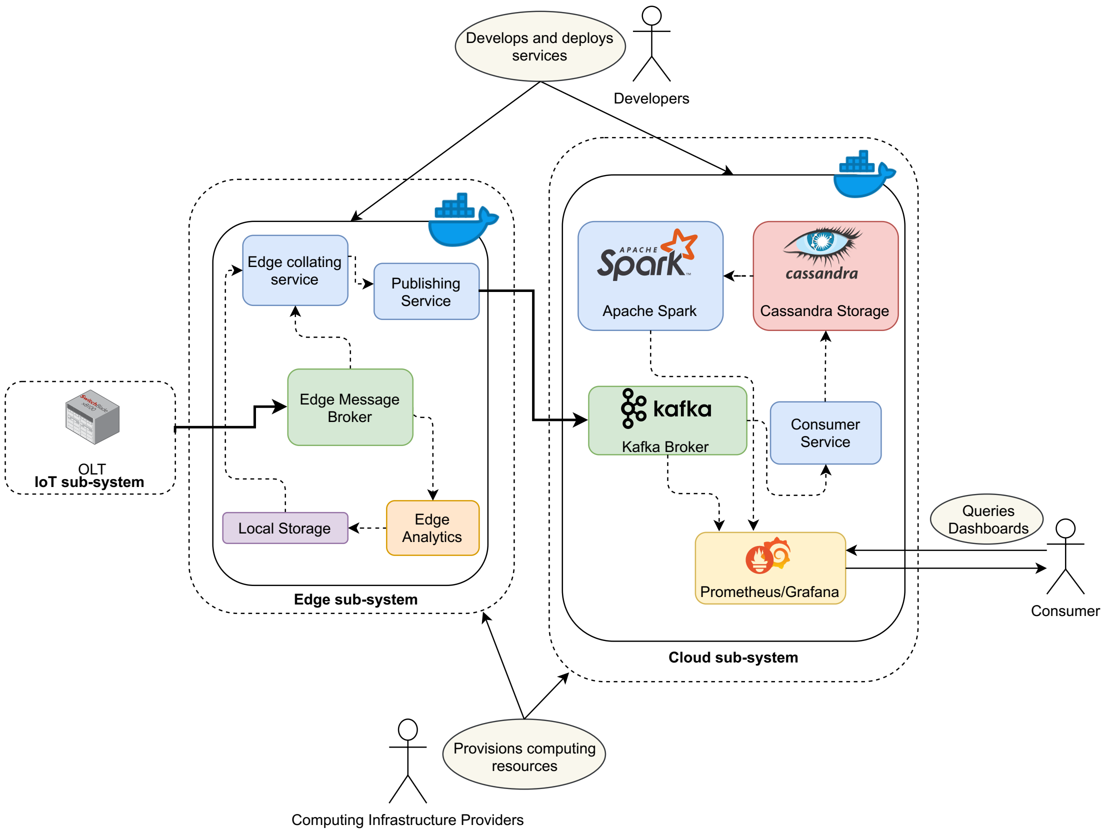


Figure 3.2: Services, interactions and stakeholders in the GPON monitoring scenario

responsible for deploying these platform services on respective infrastructure services. That is, they also fulfil the role of DevOps.

- **Cloud Service Provider (CSP)**: They provide the necessary resources upon which the developers build and deploy their platform services. The infrastructure service can be provided as IaaS, PaaS, SaaS [62] depending on the developers requirements.

- **Consumers**: They are the GPON users i.e. the ones responsible for deploying and maintaining the IoT subsystem. In the case of GPON scenario, they are the ISPs using the IoT devices and using the monitoring infrastructure on the cloud and edge subsystems.

Figure 3.2 shows a high-level representation of the various stakeholders, services and interactions in the GPON monitoring system.

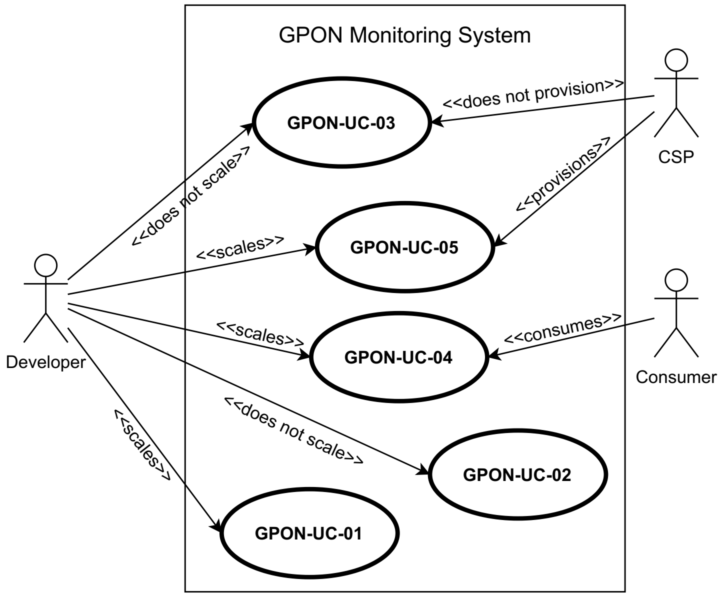


Figure 3.3: Use-cases in the GPON monitoring systems

3.2.3 Elasticity needs and ZTA approach

We briefly outlined the elasticity needs of the scenario in Section 3.2.1.1. In the GPON scenario, the elasticity needs are driven to maintain an acceptable level of all the monitoring platform services. For example, to maintain the QoS for the consumer, the IIoT developer could request to scale-up the edge brokers. If there are not enough requests, the IIoT developer can scale down the number of brokers while maintaining SLA of latency required by the consumer.

In this scenario, through the use of ZTA, the elasticity operation can be policed through a centralized PEP by stakeholders. That is, the elasticity operations of all the infrastructure services (docker or VM) in all the subsystems will go through PEP and will only be allowed if trusted. This gives the advantage of *a)* increase trust by denying implicit elasticity at all points and *b)* reduce domain expert's intervention at multiple locations thereby ensuring easier SLA compliance.

3.2.4 GPON monitoring system use-cases

We present a few use-cases in the GPON monitoring system, and its interaction with the framework. Through these use-cases, we attempt to gather requirements of a ZTA based framework that supports docker container and VM type infrastructure service. We also delineate the stakeholder and alternate scenario for a proper understanding. Figure 3.3 presents an use-case diagram of the use-cases described for the GPON scenario.

ID	GPON-UC-01
Title	Cloud message broker scale-up request
Stakeholders	Developers
Problem Statement	The cloud broker wants to request additional resources due to edge broker poisoning
Event Type	Malicious Event
Pre-conditions	• Unsecured edge broker has been attacked by a malicious entity • The attacker is publishing random values into the GPON monitoring at edge broker
Activities	• The message broker at edge requests additional instance of vertical resource such as memory to the cloud infrastructure • The elasticity request goes to a centralized resource policy enforcement point • The policy enforcement point checks the load on edge broker and the number of sensors. • The policy enforcement point denies the request
End-state	• The request of cloud broker to get more resources is denied
Alternative Activities	• We model the total expected memory required based on expert experience • Fix this maximum allocated container memory using "`--memory=*`" flag

Table 3.2: GPON use-case *GPON-UC-01*

It can be seen that the alternate flow scenario prevents altering memory allocation rules dynamically and that this memory allocation is not based on any security policy.

ID	GPON-UC-02
Title	Edge message broker scale-up request
Stakeholders	Developers
Problem Statement	The edge broker requests additional resources
Event Type	Benign Event
Pre-conditions	• There has been an increase in GPON sensor publishing rate during peak times
Activities	• The message broker at edge requests additional instance of horizontal resource that is, an extra container instance • The elasticity request goes to a centralized resource policy enforcement point • The policy enforcement point checks the time of request and number of sensors connected with the security policies. • The policy enforcement point accepts the request
End-state	• The request of edge broker to get more resources is accepted • A new managed instance of edge broker is spawned
Alternative Activities	• We fix the total expected load on the system based on sensors and time data. • The edge nodes are provided these details while bootstrapping

Table 3.3: GPON use-case *GPON-UC-02*

In the alternative flow of this use-case, the stakeholder is unable to easily manage, share or update the security rules compared to the main scenario of using a centralized point.

ID	GPON-UC-03
Title	Cloud database scale-up request
Stakeholders	Developers and CSPs
Problem Statement	A new cloud database rack needs to be spawned
Event Type	Malicious Event
Pre-conditions	• Unsecured cloud broker has been attacked by a malicious entity • The attacker is publishing random values into the GPON monitoring at cloud broker • The NoSQL database needs to spawn a new VM instance to store all the extra data
Activities	• The cloud database requests additional instance of horizontal resource that is, an extra VM rack instance • The elasticity request goes to a centralized resource policy enforcement point • The policy enforcement point checks the number of sensors connected and total ingestion for that specific day. • The policy enforcement point denies the request
End-state	• The request of cloud database to get more VM instance rack is denied
Alternative Activities	• We configure the security policy for a VM instance update in the terraform resource provider file • The database is scaled up according to these "local" policies

Table 3.4: GPON use-case *GPON-UC-03*

ID	GPON-UC-04
Title	Cloud message consumer scale down request
Stakeholders	Developers and Consumers
Problem Statement	An instance of Kafka consumer belonging to a same `consumer-group` but different `partition` requests to scale down
Event Type	Benign Event
Pre-conditions	• Some geo-partitioned edge nodes are unable to temporarily publish onto the cloud • There is a backlog of data on on edge message broker that will be immediately published after the edge node's link goes back up.
Activities	• The cloud consumer requests to remove instance of horizontal resource • The elasticity request goes to a centralized resource policy enforcement point • The policy enforcement point checks the backlog on the edge nodes and matches with the total ingestion on an average day. • The policy enforcement point denies the request
End-state	• The request of cloud message broker to scale-down is denied
Alternative Activities	• We fix the number of cloud brokers on edge at any given time and day. • There is no dynamic elasticity due to geo-partitioning

Table 3.5: GPON use-case *GPON-UC-04*

ID	GPON-UC-05
Title	Provide additional resources to analytics service
Stakeholders	Developers and CSPs
Problem Statement	Apache Spark requests additional memory resources to maintain QoS levels
Event Type	Benign Event
Pre-conditions	• This is followed by GPON-UC-04 • Since there is flood of data after GPON-UC-04, Apache Sparks needs extra memory resources to maintain SLA level QoS guarantees.
Activities	• The cloud analytics service requests additional horizontal instances of worker nodes • The elasticity request goes to a centralized resource policy enforcement point • The policy enforcement point checks if there has been an increase in throughput from database and consumers. • The policy enforcement point accepts the request
End-state	• The request of cloud analytics service to scale-up horizontally is accepted
Alternative Activities	• We estimate the number of spark worker nodes on cloud. • We deploy the ability to scale up and down depending on the load on Apache Spark independently from edge nodes.

Table 3.6: GPON use-case *GPON-UC-05*

3.3 Scenario: ML video inference system

The second scenario is of real-time ML object detection edge-cloud system. This scenario deals with detecting the count and the types of object in a given video frame. The stakeholder can use this information for a number of purpose such as security surveillance or management of crowd. This scenario of ML offloading elasticity is popularly seen in large-scale video analytics researches such as Filterforward [12][2] and [28]. There is elasticity in offloading of inference between edge and cloud. High-accuracy, high-latency inferences are performed on cloud whereas low-accuracy, low-latency inference is done on edge server. This scenario utilizes an edge-cloud continuum with the DNN object detection models situated on both edge and cloud. In this specific scenario, TinyYOLO [64] object detection for Darknet[3] framework is used. For this thesis, we have developed the deployment manifests, k3s scheduling strategy (refer to Appendix B.1) and custom web services to interact with Darknet object detection framework.

In this thesis, we only focus on improving trust of elastic offloading operations between edge and cloud. That is, we are interested in supporting stakeholders to deal with contextual cases where untrusted operations can change the output results and the latency SLA requirements.

The method of interaction with external customers is similar to popular ML SaaS products such as Google's Vision AI[4] and Azure Vision API[5]. To perform the inference, each tenet needs to use the REST API service. The input to this service is an image whereas the output is the number and names of objects detected in this image along with the coordinates and confidence. A ML inference engine (and trained model) sits on the edge device as well as cloud. Next, we describe the different platform services and computing infrastructure services deployed on this system along with the elasticity needs and ZTA support.

3.3.1 Platform and infrastructure services

3.3.1.1 Platform services

The ML model is deployed on a cluster of edge devices and one cloud worker. Apart from inference engine, the edge cluster serves a message broker for intra-pod metadata communication, a ML data pre-processing service and an externally exposed web server for client communication. All the requests and inferences are handled by this cluster of edge devices. High-accuracy requests are offloaded to

[2] https://github.com/viscloud/ff
[3] https://github.com/pjreddie/darknet
[4] https://cloud.google.com/vision
[5] https://azure.microsoft.com/en-in/services/cognitive-services/computer-vision

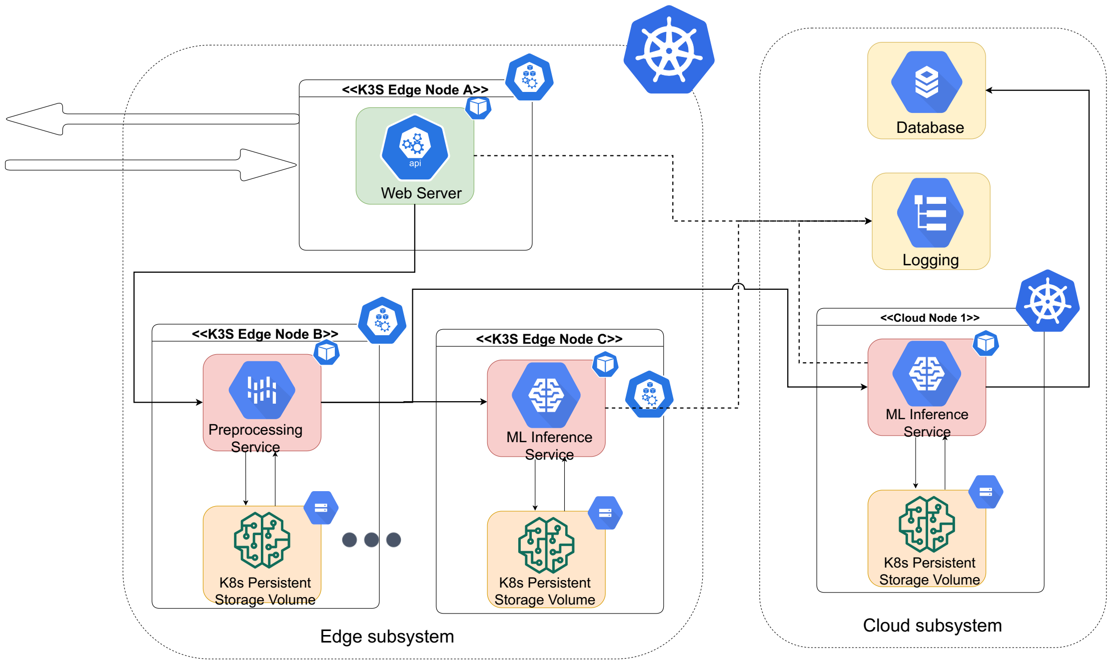


Figure 3.4: Services and interactions in the ML video inference system

cloud worker. The overall workflow can be seen in Figure 3.4. The intra-service communication within the cluster happens through appropriate REST API calls, whereas the message broker is used to check status of requests and deliver results (internally).

3.3.1.2 Deployment infrastructure

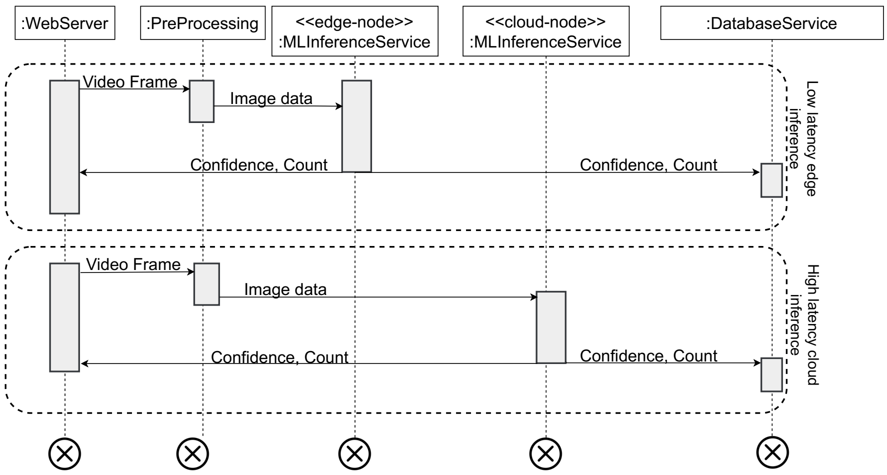
The deployment infrastructure on edge comprise of elastic state-of-the-art container orchestration framework (k3s[6]). We also have a controller node on cloud that controls these edges and cloud subsystem nodes. This scenario adds special diversity to our use-cases since unlike the GPON monitoring scenario, there is direct interaction between the various edge devices.

The various subsystems, platform services and the corresponding infrastructure services involved in this ML inference system is presented in Table 3.7.

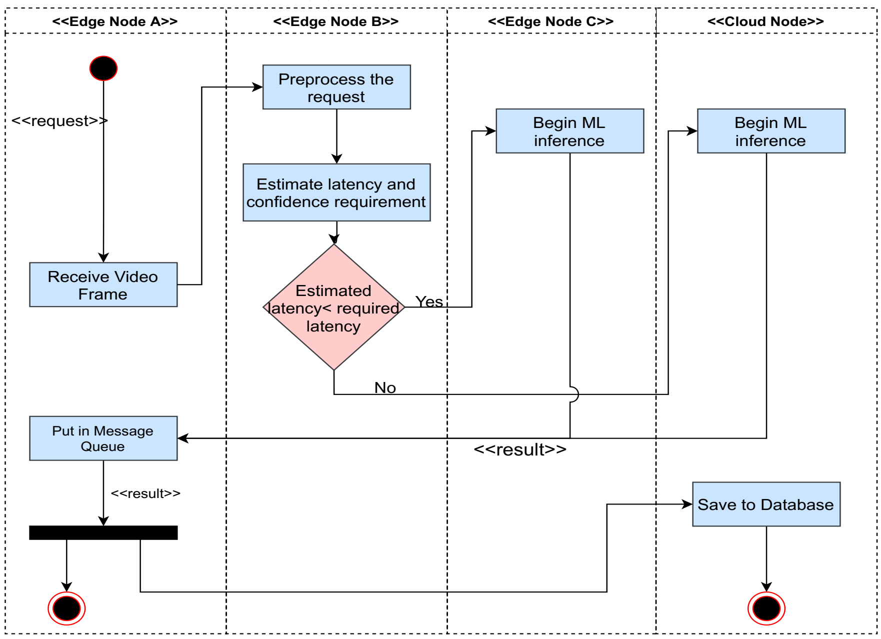
3.3.1.3 Elasticity between the services

We present the sequence of dataflow and elasticity operations in Figure 3.5. The Figure 3.5(a) presents the sequence of request (and response) flow whereas the

[6] https://k3s.io/



(a) Sequence Diagram



(b) Activity Diagram

Figure 3.5: Sequence and Activity diagram for the ML video inference system

Subsystem	Platform service units	Infrastructure service unit
Edge	Message Broker, data publishing and consuming microservices	Orchestrated container environment
Edge and Cloud	Web server, Pre-processing service, ML inference service	Orchestrated container environment (k3s and k8s)
Cloud	NoSQL database	Containers

Table 3.7: Services and elastic infrastructures in the ML video inference scenario

activity diagram in Figure 3.5(b) shows the decision-making behind the elasticity operations at various nodes.

It can be seen from the activity diagram that the edge nodes can either offload inference requests to another (stronger) edge node. Additionally, they can offload the requests to cloud subsystem if they determine that even heavier inference computation is required. Here the elasticity operations therefore include elastic offloading as well as horizontal and vertical scaling.

3.3.2 Stakeholders

We have identified the following three stakeholders in the ML video inference system. While they are similar to ones we discussed in GPON scenario, their roles, stakes, and interactions are different.

- **Developers**: They are responsible for creating and maintaining the platform services across the edge and cloud subsystem including the ML inference services. Additionally, they are also responsible for deploying these platform services. That is, they also fulfil the role of DevOps. Since our scenario focuses on an inference platform only, we assume our developers receive a pre-trained ML model and just deploy them to appropriate edge and cloud nodes.

- **Cloud Service Provider (CSP)**: They provide that necessary resources upon which the developers build and deploy their services. In this case, this infrastructure is provided as a node and pod managed container orchestration cluster of k3s.

- **Consumers**: They are the ML inference service users i.e. the ones responsible actually consuming the external APIs. They are typically under SLA agreements with the developer stakeholders that govern the latency, QoS accuracy guarantees.

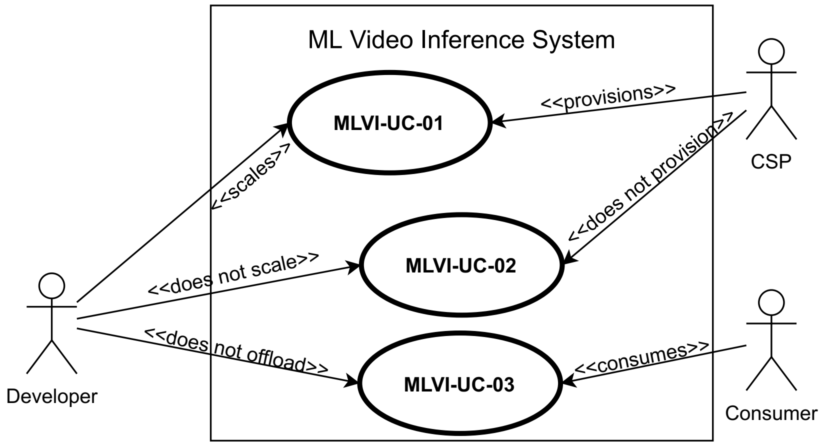


Figure 3.6: Use-cases in the ML video inference systems

3.3.3 ML video inference system use-cases

Through the use of our use-cases, we attempt to gather requirements of our ZTA based framework. We also delineate the stakeholder and alternate scenario for a proper understanding. Figure 3.6 presents an use-case diagram of the use-cases described for the ML video inference scenario.

ID	MLVI-UC-01
Title	Provide extra pods for ML inference service on edge
Stakeholders	Developers and CSP
Problem Statement	The ML inference service requests for additional instances of pods from the k3s environment
Event Type	Benign Event
Pre-conditions	• This is the most basic case • The heuristic has determined the inference be performed on edge subsystem • There is a peak load and to maintain the desired QoS, the ML inference service asks for instances of edge pods.
Activities	• The edge inference service requests additional pods on edge worker k3s nodes • The elasticity request goes to a centralized resource policy enforcement point • The policy enforcement point checks if there has been an increase in requests on the web-server and pre-processor. • The policy enforcement point accepts the request
End-state	• The request of ML inference service to scale-up horizontally is accepted
Alternative Activities	• Enable the kubernetes Horizontal Pod Autoscaler (HPA). • While deploying the ML inference service, provide the topology and maximum permissible load on the pod.

Table 3.8: ML video inference use-case *MLVI-UC-01*

ID	MLVI-UC-02
Title	Prevent forceful inference offloading
Stakeholders	Developers and CSP
Problem Statement	The attacker sends malicious data for inference and increase computation cost
Event Type	Malicious Event
Pre-conditions	• During night-time, the attacker sends specially crafted videos to increase computation • Due to the specially crafted data, heuristic has falsely identified the inference be offloaded to cloud subsystem
Activities	• The edge inference service requests to offload inference request onto already running pods on cloud worker k3s node • The elasticity request goes to a centralized resource policy enforcement point • The policy enforcement point checks the time of request. During night the probability of large human gathering is low. • The policy enforcement point rejects the request
End-state	• The request of edge ML inference service to offload the work on cloud is rejected
Alternative Activities	• Fix the time duration during which the edge can request offloading onto cloud.

Table 3.9: ML video inference use-case *MLVI-UC-02*

ID	MLVI-UC-03
Title	Prevent offloading despite consumer's elasticity SLA violation
Stakeholders	Consumers and developers
Problem Statement	The ML inference on edge does not offload the data to the cloud
Event Type	Benign Event
Pre-conditions	• The consumer's requests higher accuracy inference • The inference engine request the inference to be offloaded onto the cloud.
Activities	• The edge inference service requests to offload inference request onto already running pods on cloud worker k3s node • The elasticity request goes to a centralized resource policy enforcement point • The policy enforcement point checks the time of request. During night the probability of large human gathering is low. • The policy enforcement point rejects the request
End-state	• The request of edge ML inference service to offload the work on cloud is rejected resulting in an lower accuracy than agreed in the SLA
Alternative Activities	• Fix the priority of request allowed to every tenet to maintain the SLA level accuracy to the consumer.

Table 3.10: ML video inference use-case *MLVI-UC-03*

This is one such use-case where the policy enforcement point takes a wrong decision resulting in an lower accuracy than agreed with the consumer.

3.4 Requirements

In this section, we gather and extract the requirements from the use-cases of the two scenarios presented above. Through these requirements, we will quantify concrete functional and non-functional requirements that our framework must support. This will ensure that the models (such as data model, process model and the threat models) that we design and implement for ZETA are accurate and represent a real-world functionally usable product. It will also provide a baseline against which the final framework can be evaluated.

We present complete and singular requirements through a bunch of requirement tables. We will then proceed to indicate how each of these requirements satisfy one or more of the use-cases mentioned above.

3.4.1 Requirement tables

The requirements syntax and definitions are used as per the latest approved ISO–29148-2018 §5.2.1[1]. Each of these requirements table has an unique requirement ID, the title of requirement, a short explanation of requirement, the type (functional/non-functional) and the requirements that it depends on.

ID	RQ-01
Title	Elasticity enforcement via PEP
Description	All the elasticity operations of the system shall only take place after they have been approved by the PEP of the framework. It ensures the ZTA compliance of the system through the use of our framework.
Type	Functional
Depends on	–

Table 3.11: Elasticity enforcement via PEP *RQ-01*

ID	RQ-02
Title	Request security
Description	The ZETA framework shall ensure that all the requests from the system's services are authenticated, encrypted and non-tampered (integrity). That is, the elasticity requests coming to and the response going out to the services from the ZETA framework must follow the three basic Confidentiality, Integrity and Availability (CIA) goals.
Type	Functional
Depends on	–

Table 3.12: Request authentication requirement *RQ-02*

ID	RQ-03
Title	Rule-based authorization
Description	The ZETA framework shall ensure that the elasticity requests are approved/denied based on rules. The policy administrator shall first perform rule matching and then approve/deny the request based on the result of the rule matching.
Type	Functional
Depends on	–

Table 3.13: Rule-based authorization *RQ-03*

ID	RQ-04
Title	Framework data model
Description	The ZETA framework and the tokens shall follow predefined data models. This must include the request data model that a service uses, and the rule data model that the policy administrator and stakeholders must use.
Type	Functional
Depends on	RQ-02, RQ-03

Table 3.14: Framework data model requirement *RQ-04*

ID	RQ-05
Title	Transferable elasticity capabilities
Description	The framework should be platform agnostic and must allow the an infrastructure service to request elastic capabilities on behalf of another infrastructure service. The framework's techniques must ensure trust even after transfer of elastic capabilities.
Type	Functional
Depends on	–

Table 3.15: Transferable elasticity capabilities requirement *RQ-05*

ID	RQ-06
Title	Computing trust on context capture
Description	Computing the trust must take into consideration the context of the elasticity request as well as other (monitoring) metrics captured and saved in the framework
Type	Functional
Depends on	RQ-03, RQ-04, RQ-05

Table 3.16: Contextual trust-level computation requirement *RQ-06*

ID	RQ-07
Title	Dynamic policy administrator reconfiguration
Description	The rules defined in the policy administrator of the ZETA framework shall be configurable at the runtime. That is, the stakeholder must be able configure the rules and trust levels without the need to restart the framework.
Type	Non-Functional
Depends on	RQ-03, RQ-05

Table 3.17: Dynamic policy administrator reconfiguration requirement *RQ-07*

ID	RQ-08
Title	Policy administrator rule customizable
Description	The rules defined in the policy administrator of the ZETA framework shall be customizable at the runtime. That is, the stakeholder must be able customize the rules and add custom parameters on top of existing ones. The stakeholder may be able to change the priority and hierarchy of the rule.
Type	Non-Functional
Depends on	RQ-04, RQ-07

Table 3.18: Policy administrator rule customizable requirement *RQ-08*

ID	RQ-09
Title	Framework availability and distributed
Description	The framework shall be available and distributed to provide fault-tolerance. The latency of the response must also be in order of seconds.
Type	Non-Functional
Depends on	RQ-01, RQ-03

Table 3.19: Framework availability and latency *RQ-09*

From the use-cases, we analyse a total of nine requirements. Out of these nine, six are functional requirements and are associated with all the use-cases. These functional requirements (RQ-01 to RQ-06) form the basis of the framework and define the scope of the operations and interactions with the framework. They can be further classified as two types. The first ones improve the trust between service and framework and include RQ-01, RQ-02, RQ-03 and RQ-06. For example, RQ-02 ensures that the requests to scale up the message broker in the GPON scenario is secure and therefore increases trust between the microservice and our framework. Other functional requirement such as add structure to the framework and include RQ-04 and RQ-05. RQ-04 guarantees that all the data, requests and interaction follow a standard model across all the infrastructure services.

There are three non-functional requirements that we have identified. They are features that the framework must support to a) improve the ease of use, and b) provide flexibility and support a wider variety of infrastructure services through customizations.

Table 3.20 shows the requirement association with the use-cases. The first six functional requirements are associated with all use-cases. It is important to

	RQ-01	*RQ-02*	*RQ-03*	*RQ-04*	*RQ-05*	*RQ-06*	*RQ-07*	*RQ-08*	*RQ-09*
GPON-UC-01	✓	✓	✓	✓	✓	✓			✓
GPON-UC-02	✓	✓	✓	✓	✓	✓		✓	
GPON-UC-03	✓	✓	✓	✓	✓	✓	✓	✓	✓
GPON-UC-04	✓	✓	✓	✓	✓	✓			✓
GPON-UC-05	✓	✓	✓	✓	✓	✓	✓	✓	
MLVI-UC-01	✓	✓	✓	✓	✓	✓			
MLVI-UC-02	✓	✓	✓	✓	✓	✓		✓	✓
MLVI-UC-03	✓	✓	✓	✓	✓	✓		✓	

Table 3.20: Requirement association with use-case

note here are we have highlighted only strong association of a use-case with a requirement in the Table 3.20. For example, RQ-08 is strongly associated with use-case MLVI-UC-02 as the stakeholder will need a custom field component (time) in their rules. Whereas for MLVI-UC-01 most of the rules will not require a custom field (although some may!) and hence there is no strong association.

Through the ZETA framework along with our proposed interaction model, we support requirements from RQ-01 to RQ-08. That is, we support **all** the requirements except RQ-09. We partially support RQ-09 by providing low-latency guarantees but not availability guarantees.

3.5 Summary

In this chapter, we introduced two real-world scenarios and presented their stakeholders, types of deployment infrastructure and use-cases. All these use-cases require *some* element of zero-trust based framework to ensure an improved degree of trust in elasticity operations. Through these use-cases, we defined and analysed the functional and non-functional requirements of our framework. These requirements ensure a define constraint onto our framework to solve "real" problems while maintaining applicability to a wide variety of services and scenarios.

Chapter 4

Models, design and implementation

4.1 Overview

In the previous chapter (Chapter 3) we delineated the functional and non-functional requirements of the ZETA framework. To increase the trust of elasticity operations in edge-cloud microservices and utilize the concept of ZTA, the design of our framework must fulfil these requirements.

This chapter presents the design of our framework and describes the components involved. The interaction models in this chapter describe how the infrastructure services can use our framework. We formally present the models of trust and elasticity that we use in our edge-cloud scenarios. We also define the scope of elasticity and trust. Additionally, we describe the internal models such as the data model, evaluation model, etc. It introduces the workflow of interaction between an infrastructure service and the framework. Finally, we provide the implementation details and the choice of technologies involved in the framework.

4.2 Model of elasticity and trust

4.2.1 Model of elasticity

Our work in this thesis will focus on the *resource elasticity* and elasticity is always measured in terms of resources. All the requirements of our framework and the various use-cases will revolve around resource elasticity. The level of elasticity will focus on service units (refer to Section 2.2.1.2). Hence, the elasticity will be measured in resource allocated to individual units. An example of resource elasticity at service unit level is the variation of memory allocation to a message broker microservice.

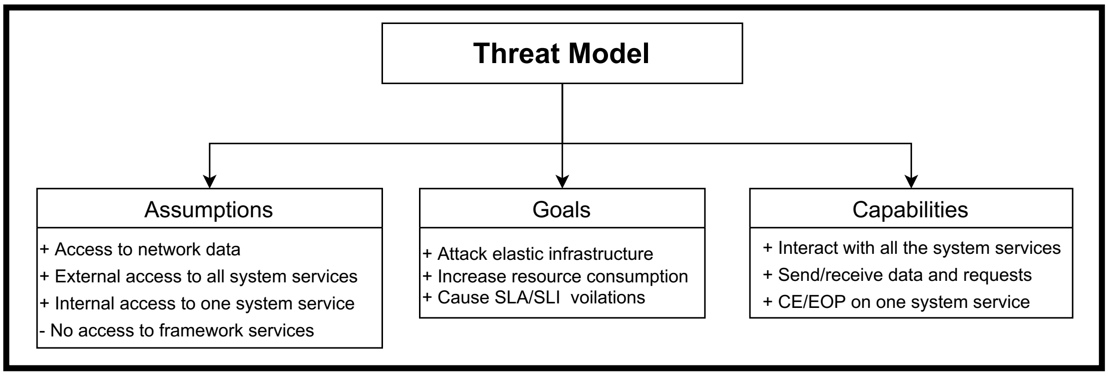


Figure 4.1: Threat model for the ZETA framework

4.2.2 Model of trust

Our work in this thesis revolves around *explicit trust* in the elasticity operations. That is, the determination of trust for *trusted-elasticity* is done through a trusted third-party. Through *trusted-elasticity*, we attempt to increase the trust in the resource elasticity requests in edge-cloud platform services. Since our choice of increasing trust is explicit, we attempt to achieve it by creating a trusted third-party framework for the validation of resource elasticity.

In our model of *trusted-elasticity*, the provider (in most cases it will be the CSP) of ZETA defines three trust level boundaries which are `HIGH`, `MEDIUM` and `LOW`. These trust levels can be controlled at run-time by the provider through trust configurations. On the other hand, the trust requirement of every platform service can be granularly customized by the tenet through configuration data and policies provided to the ZETA framework. However, they are bound to one of the three trust levels. For example, the provider can define a trust score of greater than 80 as `HIGH`. Whereas, a tenet's policy specifies that platform service X requires `HIGH` trust level. Therefore, coordination between provider and user is required.

4.3 Threat model

The threat or adversary model provides the constraints under which an attacker operates. This helps in extracting security requirements and performing evaluation of the framework's design. A threat model provides the assumptions under which an attacker operates, the attacker's capabilities (in terms of resources, attacks, etc.), actions, results, and their goals. Figure 4.1 presents the threat model under which the ZETA framework operates.

Under this threat model, the attacker has access to **all** the encrypted network data between the edge-cloud services and ZETA. The attacker can also interact freely with **all** the external (client-facing) platform services. Under this threat mode, the attacker has complete access to exactly one service and can send elasticity requests (for itself or other platform services) to the framework. Furthermore, using this "compromised" service, they can interact with all the other services. This includes sending and receiving data. They can cause arbitrary Arbitrary Code Execution (ACE) and Escalation Of Privilege (EOP) attacks on this "compromised" service.

Our threat model further assumes that ZETA component services are completely secure and the attacker does not have any internal access to framework services and can only interact with it using the two public endpoints. These endpoints are generally used by the infrastructure services to send and receive elasticity operations. Finally, the attacker is malicious and aims to increase resource consumption and disrupt the SLA/SLI of the infrastructure service. Since, this thesis focuses on using ZTA design paradigm for the microservices, our framework is designed to minimize the effects on elasticity due to lateral movement and attack propagation. Hence, such an attack model is reasonable and inline with popular ZTA threat models[1]. Ability to interact with all the external platform services and access to one internal platform service is required to validate effectiveness of ZTA.

4.4 Design

The framework comprises a collection of components. Figure 4.2 presents a high-level overview of the components inside the framework. These components are implemented as microservices. Through these microservices, the ZETA framework adds ZTA support to distributed platform services.

The framework itself supports communication with platform-agnostic infrastructure services. That is, the elasticity request of a platform service can originate from typical elastic orchestration platforms (such as Kubernetes, Docker[2], Openshift[3], Terraform[4]). The framework has three externally exposed services and four internal services. Since, our aim is to handle platform-agnostic requests, all the external requests are served using REST APIs. One of the cornerstones of the framework (as well as a functional requirement) is enforcing strong security prim-

[1]The official NIST special publication on ZTA at [67] discusses the extent of attacker access/capabilities when talking about ZTA security models. It states: "*Zero trust security models assume that an attacker is present in the environment and that an enterprise-owned environment is no different – or no more trustworthy – than any non-enterprise owned environment*"

[2]https://docker.com

[3]https://openshift.com

[4]https://terraform.io

itives (authentication & authorization) in our elasticity requests and thus all the requests require presenting a valid JWT token over TLS. To receive a token, valid credentials need to be provided by any infrastructure service. Using JWT ensures a wider availability and acceptance of the framework.

4.4.1 Token design

We propose using token based techniques for modelling the interaction between the elasticity of the platform services. Similar token based approach has been taken for authorization in extremely popular traditional frameworks such as the Kerberos [52] and OAuth [37]; and more recently in the "Update Framework" [13]. The auth and elasticity tokens are important component of our interactions. Both of them are JWT tokens and hence, are easy to use and verify. They are signed using ES256 algorithm and hence are lightweight to generate and transport. The `jti` field is used to prevent replay attacks on the elasticity tokens.

- **Auth tokens:** They are long-lived token. That is, their lifespan (expiry) is in order of hours. ZETA generates an auth token with maximum expiry time as 24 hours. Although the elasticity of the system does not depend on auth tokens, the time limit is done to mitigate long-duration token passing attacks. A sample auth token can be seen in Listing 4.1.

- **Elasticity tokens:** They are short-lived tokens with the expiration limits in the order of minutes. A constraint of 60 minutes is enforced by the framework on these tokens. The structure of the elasticity token is similar to auth token except it has `sub` field as the service using the elasticity token (deployment B from the previous example) and `aud` field as the recipient of the elasticity token (deployment A from the previous example).

It is important to note that the auth token may constitute of multiple elastic capabilities, however the elasticity token will be generated for only one capability at a time.

Listing 4.1: Sample auth token

```
{
  'sub': 'docker_service', # subject of the platform service
  'iss': 'zeta', # issuer
  'iat': 1617239022, # issed at time
  'exp': 1617245122, # expiry time
  'jti': 'fd5r23jKLd5gdf66sdYI9', # JWT ID: unique identifier
  'capabilities': ['cpu', 'memory'],
  'type': 'auth' # token type
}
```

4.4.2 Components

In this section, we briefly describe the capabilities, functions and design of all the components of ZETA framework.

4.4.2.1 Authentication endpoint

Authentication endpoint is one of the two externally exposed components of the framework and it provides the gateway for the infrastructure services to interact with the framework. It is mainly responsible for identification of the infrastructure service from where the request is originating. Moreover, any infrastructure service can request for auth token and elasticity token by providing their credentials to this component. After validation of infrastructure service credentials, it communicates with the auth component and replies back with the auth token or the elasticity token.

Any infrastructure service willing use the framework must first provide the required capabilities and the own credentials in the REST request. As per the basic HTTP authentication scheme [65], the credentials need to be provided as '`base64(username:password)`' in the body of the request. Given the modular design of the framework, support for dedicated certificates can be easily added for an easier integration with sidecar services. Auth and elasticity token requests can be seen in Listing 4.3 and Listing 4.4.

Authentication endpoint then validates the credentials from the database and passes the request to the auth component for token generation. The infrastructure services requiring the elasticity token need to provide their own credentials and also the auth token of other platform service. This is done to prevent the malicious services from requesting auth token on behalf of other platform service.

4.4.2.2 Authorization endpoint

The Authorization endpoint component is responsible for authorizing service and generating all the tokens. Additionally, it is also responsible for managing and verifying the token authenticity requests. It is important to note here that it is only responsible for authorization of capabilities and not validating the authenticity of requests (which is done by the Authentication endpoint above). To generate the elasticity token, it communicates with Policy evaluation component, Knowledge component, and Trust computation component.

The tokens themselves can be validated by any infrastructure/platform service as they are generated through elliptic curve asymmetric (public-private) key. To ensure that the resulting signature is lightweight and equally secure, we use 256-bit elliptic curve instead of 2048 bit RSA pair. Section 4.4.1 discusses the token's input parameters.

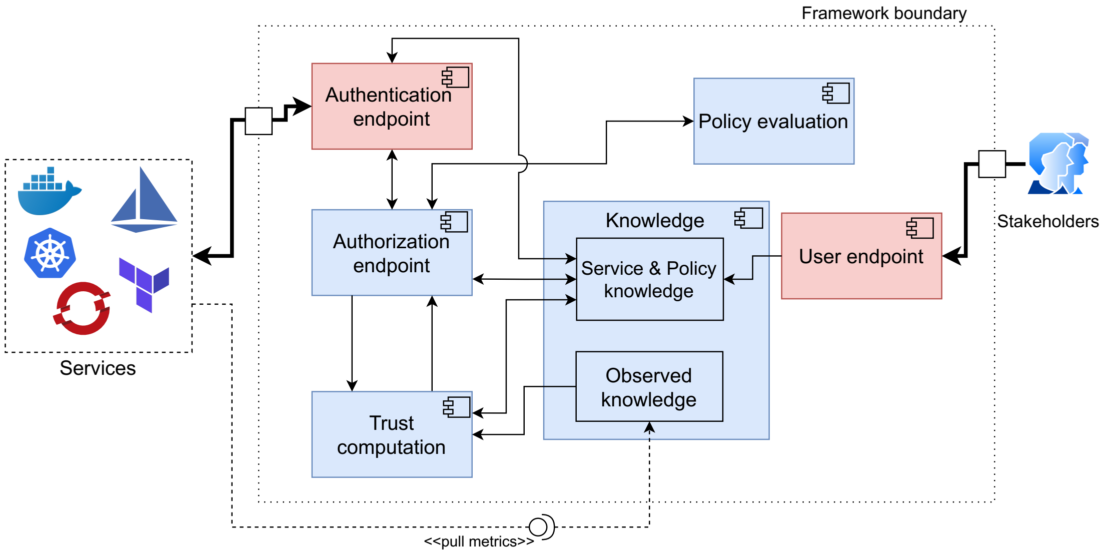


Figure 4.2: Components of the ZETA framework

This component first receives the request from service identification component. To generate the tokens, it first verifies the data from the database service, computes trust and policies by calling other components (in case of elasticity token generation) and finally generates the token. We explain the token generation workflow in detail in the Section 4.6.

4.4.2.3 Knowledge component

The Knowledge component acts as a central repository for the knowledge about the platform service capabilities, tokens issued, rules uploaded and trust evaluation. Moreover, it also has monitoring metrics of these platform services. This knowledge provides the **context** for trust evaluation and is used by the trust evaluation component.

This component contains two types of contextual knowledge. This approach of capturing knowledge is taken from [53] that presents the BeyondCorp zero-trust framework.

- *Service and Policy knowledge*: This provides the context of the platform services that are involved in the elasticity request. They could be the services that are requesting the elasticity or the object of elasticity. For example, if service X and Y are both deployed on same infrastructure, this is one of the context that trust evaluation component may use. All the service identification (credentials) and service authorization (tokens) are also saved as service knowledge. This knowledge is saved as relational knowledge.

- *Observed knowledge*: This provides contextual knowledge into the status of the platform services as well as the infrastucture services, and forms a very important aspect of the trust evaluation component. For example, in case of an elasticity request to increase CPU resource, the trust evaluation component checks the current CPU usage knowledge of the target platform service and assigns a low trust score if the current CPU usage is low. The observed knowledge is pulled as telemetry information and are saved as temporal knowledge in database.

4.4.2.4 Policy evaluation component

Policy evaluation component allows the developers to define custom policies. Often these policies are specific for an organization and hence, this component adds flexibility in defining custom organization wide rules. While the trust computation level is defined by the provider, a policy defines how a stakeholder uses the trust level for their platform service. The input to this component is the rule and the data uploaded by the stakeholder as well as the trust evaluation result. It is called by the Authorization endpoint component and is the final point of request evaluation before the elasticity token is granted.

From our motivating use-cases, allowing the developers to define custom rules is one of functional requirements of the framework. The rules and related data are uploaded into the policy knowledge by the stakeholder. We use `Rego`[5] based rules and JSON based data in the Policy evaluation component. `Rego` supports structured JSON document models and integrates well with popular policy evaluation engines. It is declarative language used for defining policies for microservices such as [57].

4.4.2.5 User endpoint

This component is responsible for handling the stakeholder requests (such as rules upload) and managing them into the knowledge component. It provides an interface to the stakeholder in customizing their rules and selecting their trust models. All the data dealt by this component is managed in the service and policy knowledge context only.

To ensure the ease of use, this component exposes multiple RESTful endpoints that allows the stakeholder to manage their polices. It relies on basic but trusted and popular security primitives such as HTTP1.1 “authorization header” for authentication of the users [65].

[5] https://www.openpolicyagent.org/docs/latest/policy-language/

4.5 Trust computation component

The Trust computation component is a central component of the framework. The Trust computation component computes a confidence score (and subsequently trust level) from the contextual knowledge (service and observed) gathered in the Knowledge component. This trust level is used in the Policy evaluation component to determine the authorization based on the rule. For example, a stakeholder may define a rule to authorize elasticity if the trust level is above a certain threshold. This component also provides extensibility to our framework. That is, it allows the stakeholder (provider in this case) to replace different trust computation algorithms depending on their requirements. Moreover, through configurations, the provider can also customize the trust level and adds the flexibility feature of the framework.

The trust models scores the confidence values of a request on a scale of 0-100. This value is then changed to a trust level by the component. These trust levels are `LOW`, `MEDIUM`, and `HIGH`. We perform this score to level mapping from the table shown in Table 4.1 which is derived from STIX 2.1 OpenCTI model [19].

It is important to note that the trust computation component only outputs the trust level from the Table 4.1, whereas the policy evaluation component enforces it in the request. The default trust level is calculated using the GPR method as described in the Section 2.2.5. We delineate the exact multi-variate trust level generation mechanism from the GPR output in Section 4.5.2. However, ZETA gives complete flexibility to the provider to change the trust parameters and algorithms. We discuss these techniques in subsequent sections.

4.5.1 Trust parameters and algorithm

Figure 4.3 presents the different subcomponents and the interface of our trust component. For trust algorithm extensibility, any trust algorithm (custom or default) needs to implement the `TrustAlgorithm` interface. The default GPR algorithm already implements this interface. A custom trust algorithm can be provided by the stakeholder via the User endpoint component.

Trust Level	**Confidence score**	**Range**
None	0	0
Low	15	1-29
Med	50	30-69
High	85	70-100

Table 4.1: Trust level to confidence score

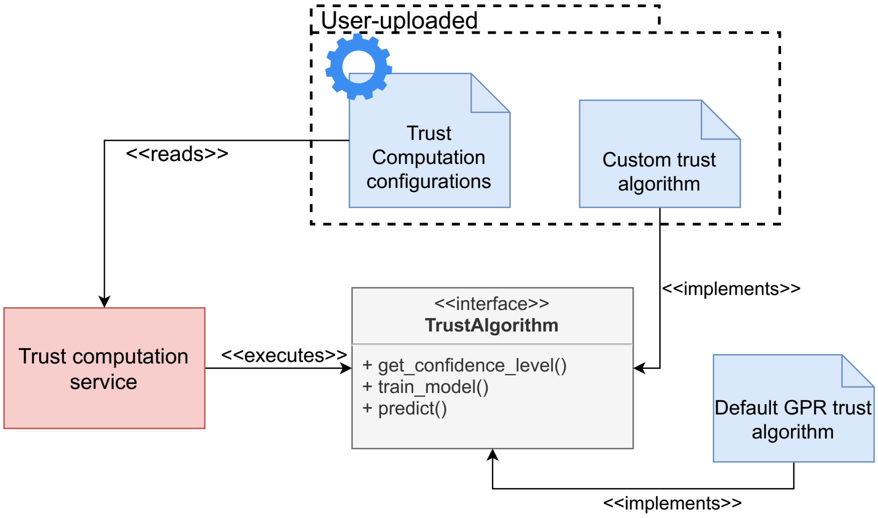


Figure 4.3: Interfaces and subcomponents of Trust computation component

The framework provides customizability via configurations. These configurations are used to control the "appropriate" level of trust. Listing 4.2 provides a sample configuration. There are two types of configurations that the stakeholder can specify *"parameters"* and *"custom"*. These configurations aid in abstracting the mathematical background required for fine-tuning the trust levels.

"parameters": They are three pre-defined values that the trust algorithm must use. a) `trust-sentivity`: this is between the range 1-10 with 1 being most strict and 10 being least strict. b) `noise`: Noise can be one of `low`, `medium` or `high` and dictates the noise introduced in the training dataset. A higher noise would mean lower trust boundaries. Finally, c) `accuracy` is also one of `low`, `medium` or `high` and dictates the rounds of optimizations used in the code. ZETA maps each of these values to a specific mathematical number. For example, `low` accuracy implies 5 rounds of optimization and `low` noise value implies 10^{-2} white noise addition. We further allow the using *"custom"* fields into the configuration that are required by the custom algorithms. They are provided to the custom trust algorithm during runtime and allows them to algorithm specific customizations. They are not required if the default classifier is used.

GPR performs non-linear multi-dimensional regression and has more recently found exploitation in multiple domains [70]. It is especially beneficial for our use-case as this process a) gives us the probabilistic uncertainties as confidence intervals and b) performs well for small observed points. Moreover, it is fast to train a model, perform a regression fit and sample values from this trained model [63]. For all these reasons, we use GPR for calculating contextual trust.

Listing 4.2: Sample trust computation configuration

```
---
trustAlgorithm: default_gpr
# Specify the model parameters
parameters:
  trust-sensitivity: 5
  noise: medium
  accuracy: high
#specify custom parameters that are specific for models
custom:
  random_key: 'random_value'
---
```

4.5.2 Trust level computation algorithm

Our approach models the uncertainties generated from the trained model into three levels *viz Low trust*, *Medium trust* and *High trust* as defined in the Table 4.1. To begin the processing, the observed values of any service are pulled from the observed knowledge base.

The first step is normalization of the values in the scale of [0-100]. We use a modified version of *min-max normalization* technique (Equation 4.1).

$$x' = 100\left(\frac{x - x_{min}}{x_{max} - x_{min}}\right) \ \forall x \in arr(x) \tag{4.1}$$

In the next step, we add some noise to the training points is to allow smaller (instead of zero) trust levels. around the observed points and prevent over-fitting of the regression curve. The mean is kept as **0** and the noise is then added to the observed points. The noise level bounds is read from `noise` parameter. The error bounds are fixed between 10^{-10} and 10^{-1}. The final kernel is thus represented by a summation of RBF and white noise kernel. Finally the GPR is done as per Equation 2.1 with RBF kernel length as 10^2 (max output from Equation 4.1). Moreover, our trust calculation component also adds a white noise kernel that can represent independent noises in the training data. This is required to prevent "tight-fitting" of the regression curves around the observed points.

$$Trust_{range} = 2^i \times t_s\left(1.96\sigma\right), \text{ where } i \in \{1, 2, 3\} \tag{4.2}$$

To calculate the trust levels, we utilize the 95% confidence interval whose Z-score is 1.96[6]. However, as the training point noise is small, the resulting output

[6] For any standard normal distribution, a variable Z falls between ± 1.96 with a 95%

Algorithm 1 Default trust computation algorithm

```
Require: x_data, trust-config.yml
Ensure: << TrustAlgorithm >> is implemented
  function ModifiedMinMaxNormalization(x_arr)
      x_norm ← arr_empty
      for all x_i ∈ x_arr do
          x_norm ← x_norm.push(100 * (x_i − x_min)/(x_max − x_min))
      end for
      return x_norm
  end function

  function GetTrustLevel(y_r, s_d)                    ▷ Generates the trust level
      t_low = 4 * t_s * (1.96 * s_d)
      t_medium = 2 * t_s * (1.96 * s_d)
      t_high = t_s * (1.96 * s_d)
      if |y_r| ≤ t_high then
          return HIGH
      end if
      if |y_r| > t_high and |y_r| ≤ t_medium then
          return MEDIUM
      end if
      if |y_r| > t_medium and |y_r| ≤ t_low then
          return LOW
      end if
      return NONE
  end function

  params ← ReadConfigParameters('trust-config.yml')
  n_s = NoiseToValue(params['noise'])                               ▷ Noise
  a_c = AccuracyToValue(params['noise'])                            ▷ Accuracy
  t_s = int(params['trust − sensitivity'])                          ▷ Trust-sensitivity
  x_norm = ModifiedMinMaxNormalization(x_data)

  kernel = RBF(100, (10^-1, 10^2)) + W_N(n_s)          ▷ From sci-kit learn package
  y_r, s_d = GPR(x_norm, kernel, a_c)          ▷ Regression from sci-kit learn package
  return GetTrustLevel(y_r, s_d)
```

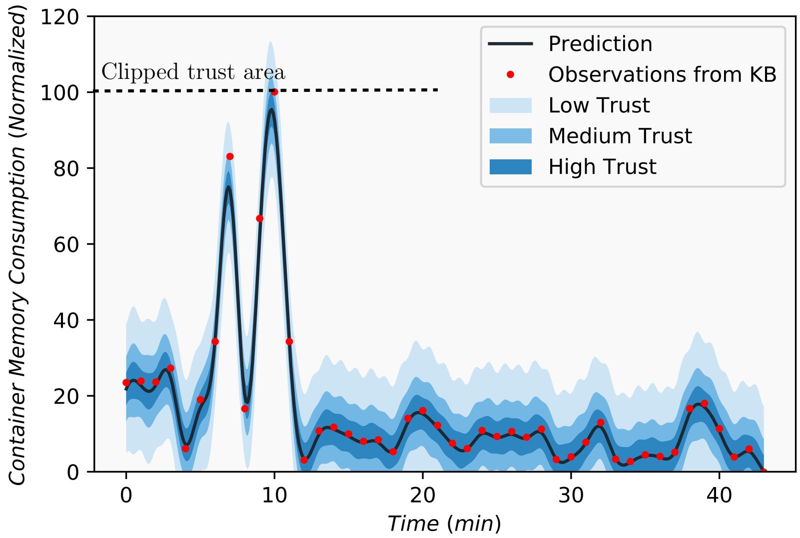


Figure 4.4: GPR regression curve and trust levels for container memory consumption

σ will also be lower. Therefore, we multiply the Confidence Interval (CI) range to a constant to define smaller trust levels. The final three trust levels are calculated as per Equation 4.2. Here the σ is sampled out from the GPR trained model and t_s is provided through the trust configuration input as `trust-senstivity`.

Algorithm 1 presents our default trust level computation algorithm for a single dimension GPR. In case of multi-dimentional GPR, a weighted average of the trust levels is taken from all the GPR trust dimensions. Equation 4.3 shows the calculation of the final trust level score where all the weights have equal values. Here each x_i represents an individual trust level generated from multi-dimensional GPR such as CPU consumption, memory consumption, etc.

$$Score_{conf} = \frac{1}{n}\sum_{i=1}^{n} x_i \tag{4.3}$$

The input to the default trust algorithm is time and resource. Prior to regression, the resource usage is normalized as 0–100 (0 representing the minimum resource utilization and 100 representing maximum resource utilization) through the `ModifiedMinMaxNormalization` function in Algorithm 1. All the values above 100 are clipped. Figure 4.4 visualizes a sample single dimension GPR and the corresponding trust levels for a docker container memory utilization from the ML

probability. Hence, this is also known as 95% CI Z-score. For more information refer: https://www1.udel.edu/htr/Statistics/Notes/class13.html video inference scenario (For the details on the testbed, elasticity conditions and testing criteria refer to Section 5.2). It shows the ranges of the three trust levels and how the confidence score would decrease when a value is sampled at a larger distance from the regression curve.

4.6 Interaction flows

Usage of our framework requires multiple workflows such as generation of tokens, observed knowledge capture and rule management by the stakeholder. This section presents the activities carried out by the framework to complete these workflows. The two more important workflows are the generation of authentication and elasticity tokens, and they require complex interaction of different components in the framework.

4.6.1 Auth token generation workflow

Generation of auth token involves the interaction between authentication and authorization components. Figure 4.5 shows the activities and interaction of these components. The steps involved are:

(a) The infrastructure service first generates a request for the auth token along with the capabilities. It also needs to provide its own credentials in this request. A sample request body can be seen in Listing 4.1.

(b) The Authentication endpoint component first validates the credentials from the Knowledge component. If the credentials don't match, an error response is sent back.

(c) The Authentication endpoint component sends the request to the Authorization endpoint component. The Authorization endpoint component checks if the capabilities by the service are allowed by the stakeholder. It is a basic check and is done directly with the knowledge component.

(d) The Authorization endpoint component generates a valid JWT auth token along with valid fields (refer to Listing 4.3) and signs it private key. This is sent back as a response to the Authentication endpoint component which returns it to the infrastructure service.

(e) The generated auth token and some metadata token (such as time of generation, service id) is *parallelly* saved into the (service) Knowledge component.

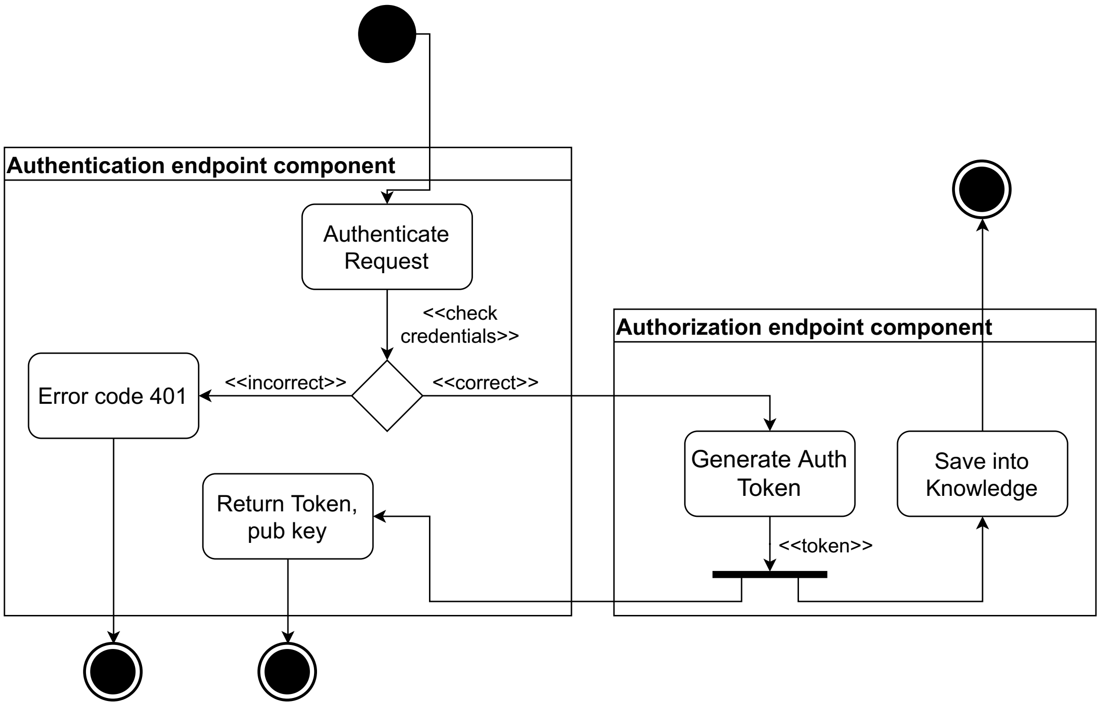


Figure 4.5: Workflow for the generation of auth token

4.6.2 Elasticity token generation workflow

Figure 4.6 presents the activities *inside* the components of the framework that occur during the elasticity token generation workflow. As described earlier, the elasticity token generation request is done by infrastructure services and they need to present the auth token to receive the elasticity token. The steps involved are:

(a) The infrastructure service generates the elasticity request and sends this request to Authentication endpoint component along with the auth token that it received from the target infrastructure service.

(b) The Authentication endpoint component checks the credentials of the infrastructure service from the Knowledge component. And similar to the auth token, if the credentials do not match, it returns an error message.

(c) Next the request is sent to Authorization endpoint component. It first validates the Authorization token (including checking for expiry etc). If there is some issue with the validation, the Authentication endpoint component returns an error message to the infrastructure service.

(d) Upon successful validation, the Authorization endpoint component sends a

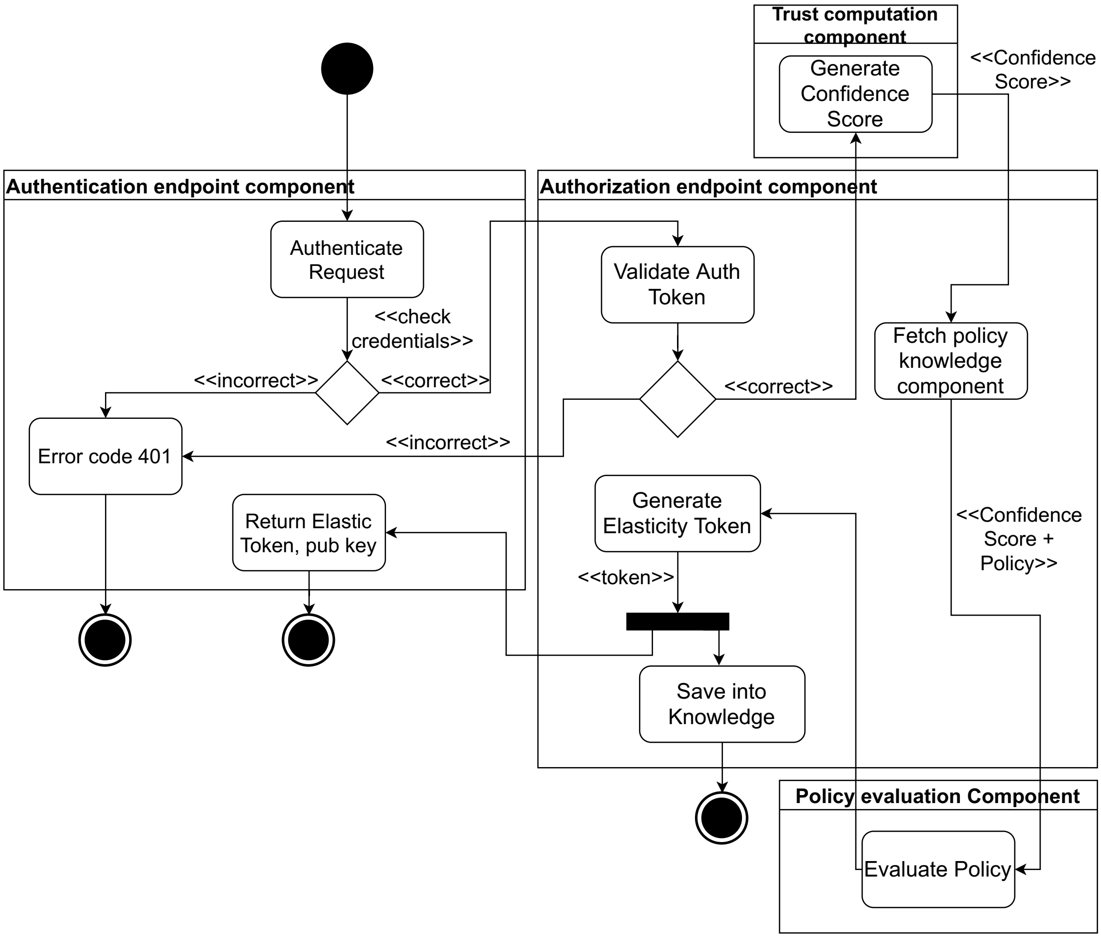


Figure 4.6: Workflow for the generation of elasticity token

request to the Trust computation component which calculates and returns a confidence score (refer to Table 4.1).

(e) The Authorization endpoint component next fetches the policy knowledge from the Knowledge component and sends a request to the Policy evaluation component along with the policy and confidence score.

(f) The Policy evaluation component evaluates the rules uploaded by the stakeholders and the confidence score and returns its decision.

(g) If the decision is negative, the Authentication endpoint component returns error response.

(h) If the policy decision is positive, the Authorization endpoint component generates a valid JWT elasticity token and with the capabilities and signs it

private key. This is sent back as a response to the Authentication endpoint component which returns it to the infrastructure service.

(i) The elasticity token is also saved into the Knowledge component for future trust evaluation.

There are two smaller workflows for the policy evaluation and confidence score generation.

- **Calculating trust**: The trust calculation component performs two activities to generate the confidence score. It first fetches the observed and service knowledge from the knowledge component. It then uses one of the appropriate GPR to calculate the score. The framework supports every trust algorithm as plugin (currently only GPR is implemented) and the stakeholder can use any one of them.

- **Evaluating policy**: The policy evaluation component is a single step process. Both the policy and the confidence score are pushed by the authorization component. The policy engine evaluates the policy and returns a boolean result.

4.6.3 Knowledge workflow

To capture context, a period-based pull monitoring technique is used. We use popular monitoring tools to capture the observed knowledge. In this monitoring technique, the monitoring tool's server (agent node), periodically fetches the metrics from the infrastructure service (monitored node). We use pull based monitoring as it provides improved security as compared to other event-driven/push based monitoring techniques [75].

The observed knowledge workflow has two aspects :

(a) An agent collects the infrastructure service metrics and logs and pushes them into the observed knowledge component.

(b) Whenever the trust component wishes to use this knowledge, it uses queries to fetch this saved data.

4.6.4 Rule management workflow

This workflow is done by the stakeholder to perform CRUD operations on the rules. As mentioned earlier, the rules are of two types, policy rules and user-specific rules. Policy rules are provided to this component as Rego policies. User-specific rules are provided as simple JSON objects.

The User endpoint component receives request, does *some* basic validations and pushes/updates/deletes the data in the Knowledge component and the response (success/failure) is returned to the user.

4.7 Interaction model

The interaction model defines the interaction of infrastructure services among each other and with the framework for elastic resource request. It also delineates the techniques that they utilize for elastic resource request with other homogenous or heterogeneous infrastructure services.

4.7.1 Interaction model – *"Removing trust from elasticity"*

Our interaction model requires that the elasticity operations only take place if the infrastructure service is presented with appropriate authorization token. This cryptographically verifiable token is issued by our framework and represents the elasticity capabilities (and scope) of a particular infrastructure service. Next, we describe three different interaction situations on how the framework (and tokens) ensures ZTA compliance with integrity in the elasticity of the infrastructure service.

4.7.1.1 Single infrastructure service (self-elasticity)

Using tokens for self-elasticity on a single infrastructure service is straightforward. First the infrastructure service gets the auth token and the verification public key from the framework that represents its elastic capabilities. Next, if the service want to scale-up or down, it will send a request for the elasticity token along with the auth token. At this point, the trust is calculated and results (elasticity token) returned. Next the service can optionally request the verification of the elasticity token before scaling up or down. Figure 4.7 shows one such example of interaction between the infrastructure service (docker) and framework.

4.7.1.2 Multiple services on a heterogeneous infrastructure

Figure 4.8 presents a sample interaction of two services (and the framework) and delegation of elastic request capabilities to another service on a different infrastructure service by a deployment (deployment A). Deployment A first receives the auth token and the public key to verify this token from the framework. This auth token is then shared with another infrastructure (deployment B) over 'data plane'.

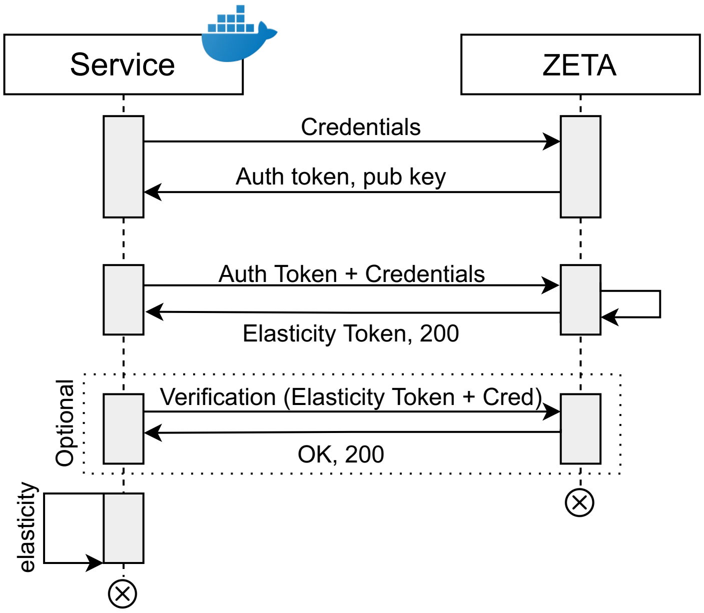


Figure 4.7: Single infrastructure service interaction with the framework

The deployment B can now present this token to the framework to receive the elasticity token for deployment A. The ZETA framework computes the trust, checks the policies and issues the elasticity token. To request elasticity in deployment A, deployment B can then send out this elasticity token to deployment A.

In accordance with the ZTA principle, the Deployment A must only scale up when it receives the "elasticity" token from Deployment B. Deployment B can get the elasticity token if it has a) the auth token from deployment A and b) gets trusted by the framework. Similar to the previous scenario, the deployment A can optionally ask the framework to verify the elasticity token before elasticity operation. This is to support use-cases where deployment A wants to verify the status of the public key (expired/revoked/rotated).

4.7.1.3 Multiple services on homogenous infrastructure

This type of interaction requires transfer of the authorization JWT token between a single but distributed infrastructure service. These services operate under same domain i.e. they may be able to provision their resources from the same system. The interaction in this case is very similar to the previous case (see Figure 4.8) however, the auth token's transfer does not occur over data plane, but over control plane. The framework's trust computation service takes this context into consideration while issuing elasticity tokens.

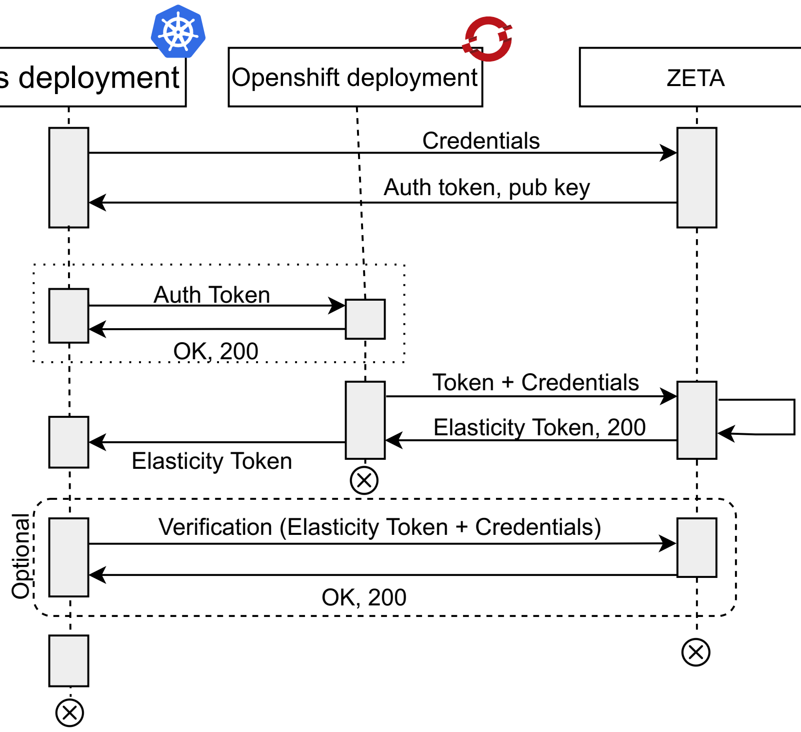


Figure 4.8: Heterogeneous deployments interaction with the framework

4.8 Implementation

This section presents the details of the framework implementation. The implementation of our framework follows a microservice architecture. Microservice architecture is based on the principle of shared-nothing and leverages the message passing mechanisms of distributed systems for communication of data [69]. Using this architectural design paradigm gives us the benefit of loosely coupled, reusable and independently deployable (and maintainable!) pieces of code [5]. In our implementation, each of the component (from Figure 4.2) is created as exactly one microservice except for the Knowledge component. The Knowledge component saves the two different types of knowledge (observed and service) into two separate microservices.

Before presenting the implementation details of the framework, some finer details of the implementation are described:

- **Programming language**: All the components have been programmed in **Python**[7] v3.8. It provides object-oriented features along with dynamically

[7] https://www.python.org

typed syntax. This significantly eliminates writing down boilerplate code during development.

- **Package management**: The framework uses **pip**[8] package manager for installation and management of python libraries.
- **Container Management**: All the components are build and deployed as containers to facilitate platform independent execution. Our framework uses **Docker** to build, install, manage and run container images of the framework.
- **Version Control**: **Git**[9] was used as a decentralized version control system.

Next, we discuss the implementation of the components in detail.

4.8.1 Authentication endpoint component

The Authentication endpoint component comprise of two endpoints. That is, `/auth` and `/elasticity` endpoints. All the requests are made as RESTful JSON objects. A sample request to `/auth` service can be seen in Listing 4.3. A corresponding `/elasticity` service request is presented in Listing 4.4. Flask REST library is used to provide the web server for services.

Listing 4.3: Sample auth token request

```
{
  'service_name': 'database_service',
  'duration': '10h',
  'credentials': 'cm9oaXQ6cGFzc3dvcmQ=',
  'capabilities_requested': [memory, cpu]
}
```

Listing 4.4: Sample elasticity token request

```
{
  'target_service_name': 'database_service',
  'service_name': 'message_broker_service',
  'duration': '30m',
  'auth_token': XYZ,
  'credentials': 'dai78Whj87687dsJK8kjQ=',
  'capabilities_requested': [memory]
}
```

[8] https://pypi.org/project/pip/
[9] https://git-scm.com/

Note that in Listing 4.4, the request duration is shorter, the `auth_token_field` is present and the capabilities are a subset of auth request.

4.8.2 Authorization endpoint component

The Authorization endpoint component generates the authorization token and elasticity token and has two endpoints that are `/auth` and `/elasticity` respectively. The `GET` requests are used to create tokens while `POST` is used to validate them. To query the service knowledge database, psycops2[10] library is used. Finally, it calls the trust computation service and policy evaluation service through REST calls.

To generate the token, this component creates the JWT claims as shown in Listing 4.5. It is important to note that auth tokens claims are automatically validated before the elasticity tokens are generated.

Listing 4.5: Python snippet to create JWT claims

```python
# Add capabilities to reflect what this JWT token can do
return {
  "exp": datetime.utcnow() + timedelta(hours=expiry),
  "nbf": datetime.utcnow(),
  "aud": [f"tefa:{service_name}"],
  "cf": claims
}
```

4.8.3 Knowledge component

The knowledge component consists of two different types of knowledge source. Therefore, there are two different services that are used for their implementation.

The *service* & *policy* knowledge consists relational data to store the tokens, services and policies. These mapping and relations between these data can be one-to-one (a tenet to a policy), or many-to-one (platform service to all the auth tokens or elasticity tokens). Hence, to save the service and policy knowledge, a relational database is preferable. In ZETA framework, PostgreSQL[11] is used. It is an open source relational database that supports Atomicity Consistency Isolation and Durability (ACID) transactions and Structured Query Language (SQL)-2016 [47]

[10] https://www.psycopg.org/docs/

[11] https://www.postgresql.org/

Listing 4.6: Sample tenet uploaded Rego policy

```
package zeta

import data.zeta_framework.client_1
default trust_client_1 = false

trust_client_1 {
  trust := input.trust
  client_1.trust == trust
  client_1.service_name == "edge-inference-server"
  input.time > 10
}
```

complaint queries. The *observed* knowledge is a time-series data and InfluxDB[12]. The telemetry data into InfluxDB can be pull via telgraf server-agent plugin.

4.8.4 Policy evaluation component

The policy evaluation component comprise a web-server for handling API requests and Open Policy Agent (OPA)[13] as the general purpose policy evaluation engine. OPA is a state-of-art and popular choice for policy evaluation in cloud-native environments. In ZETA, OPA works in `input` overload mode which is recommended for local and dynamic data. The user-endpoint component is used to add and update the stakeholder specific data. The client data bundle is partitioned on per-tenet basis and uploaded as JSON.

Policy evaluation component evaluates the policy uploaded by the user with the output from trust-evaluation component. The output from trust-evaluation component is read from overloaded input parameter whereas client specific data is fetched from the data bundle. The policy language is Rego and Listing 4.6 presents a simple rule from an uploaded policy. Listing 4.7 provides a sample tenet specific data evaluated by the policy.

[12]https://www.influxdata.com/
[13]https://www.openpolicyagent.org/docs/latest/

Listing 4.7: Sample tenet uploaded policy data

```
{
  "data": {
    "zeta_framework": {
      "client_1": {
        "trust" : "MEDIUM",
        "service_name" : "edge-inference-server"
      },
    }
  },
  "endpoint": "zeta_framework/client_1"
}
```

4.8.5 Trust computation component

The default trust computation component implements the `<<TrustAlgorithm>>` interface and is a direct implementation of the trust algorithm defined in Algorithm 1. We use Scikit-learn [56] only to determine kernel and perform GPR. As with all other components, the trust can be calculated through appropriate REST API calls on `confidence` endpoints. As per the workflow, it is triggered by the Authorization endpoint component.

4.9 Summary

This chapter takes a deep-dive into the ZETA framework. Our interaction model provides a secure way of establishing trust along with delegation of elasticity capabilities using lightweight JWT token. This interaction model supports a diverse homogenous and heterogeneous infrastructure services.

In this chapter we further delineate the components of the framework that work together to these tokens. Together with the interaction within these components, there exists a complex workflow that includes computation of trust, evaluation of policies and dataflow into knowledge bases. We further discuss the extensibility extensibility of the framework by allowing the stakeholder to implement a custom trust algorithm and fine-tune it. These configurations also aid in abstracting the mathematical background required for fine-tuning the trust. Finally, this chapter concludes with the implementation details of these components.

Chapter 5

Evaluation

5.1 Overview

This chapter presents an evaluation of the zero-trust elasticity framework designed in the previous chapters. The validation of the design is done on two real-world motivating examples presented in Sections 3.3 and 3.2. The trust configuration variation on these two contrasting scenarios validate our claim of fine-tuning and extensibility of ZETA on varied use-cases.

We further demonstrate the usefulness of the framework by performance evaluation and quantifying the end-to-end latency guarantees. Finally, we discuss the evaluation results in the two contrasting scenario and in the context of the scalability of the system. We also model adversary tactics and possible attacks on the designed framework and the interaction model; and present possible improvements on the framework.

5.1.1 Evaluation parameters

The evaluation is of two types *a)* demonstrate contextual trust computation by our algorithm and ability to fine-tune it through configurations and *b)* quantification of resource requirement and scalability by performance evaluation of the framework. The former is done to demonstrate the framework's ability to perform contextual trust computation based on high-level view of the platform services. Additionally, it also validates the major research objectives i.e. 'configurability' of trust levels. The latter type of evaluation (performance) provides real-world corroboration of the ZETA framework. That is, it helps in defining the relation between scalability and resource requirements for using it on a large-scale scenario. We later provide the interpretation of graphs and discussion of these results.

Subsystem	Type	Hardware	Quantity
Cloud	Master node	3 core 2Ghz Intel(R) Xeon(R) CPU	1
		4 GiB RAM, 25Gb/s network	
Cloud	Worker node	14 core 2Ghz Intel(R) Xeon(R) CPU	1
		112 GiB RAM, 10GB/s network	
		16GB NVIDIA Tesla P100 GPGPU	
Edge	Worker node	Raspberry Pi 4 Model B	3
		4GB RAM, 4 core 1.5Ghz CPU	

Table 5.1: Hardware profile of the ML video inference testbed

5.1.2 Deployment of ZETA framework

We deployed ZETA framework on a VM with each components in a separate docker container. The resource requirement of ZETA is low and can be easily deployed on high-end edge devices or a low-end cloud VM. Our choice was deployment a VM provided by the CSC[1]. The VM had 4 vCPU and 8GB RAM. The choice of OS was Ubuntu 20.04 and all the components of the framework were run via a synchronized `docker-compose` file. The edge devices from the ML video inference system could communicate with the framework over public IP whereas the cloud subsystem can communicate using personal “VNF”. The trust algorithm used for all the evaluation was the default GPR.

5.2 Scenario I: Elastic ML video inference

5.2.1 Testbed

The ML video inference system testbed has been implemented on state-of-art elastic technologies to model our testbed with popular computer vision products such as Azure Vision API[2]. The distributed testbed cluster comprised of multiple Raspberry PI model B[3] (as edge nodes) and cloud nodes.

The infrastructure of the testbed subsystems are provided in Table 5.1. The resources allocated to the edge and cloud subsystems are conventional low-resource edge and high-resources cloud. Our edge-cloud execution model used a proximal

[1]CSC-Tieteen tietotekniikan keskus Oy, https://csc.fi
[2]https://azure.microsoft.com/en-in/services/cognitive-services/computer-vision
[3]https://www.raspberrypi.org/products/raspberry-pi-4-model-b/

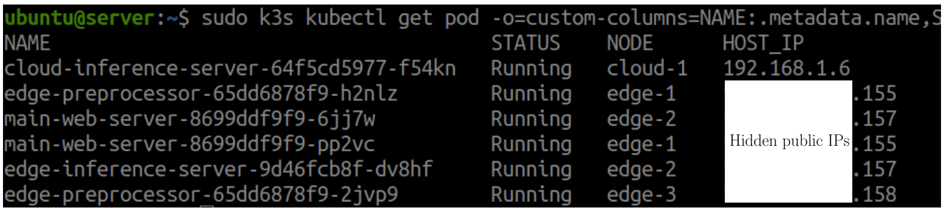


Figure 5.1: K3s node assignment

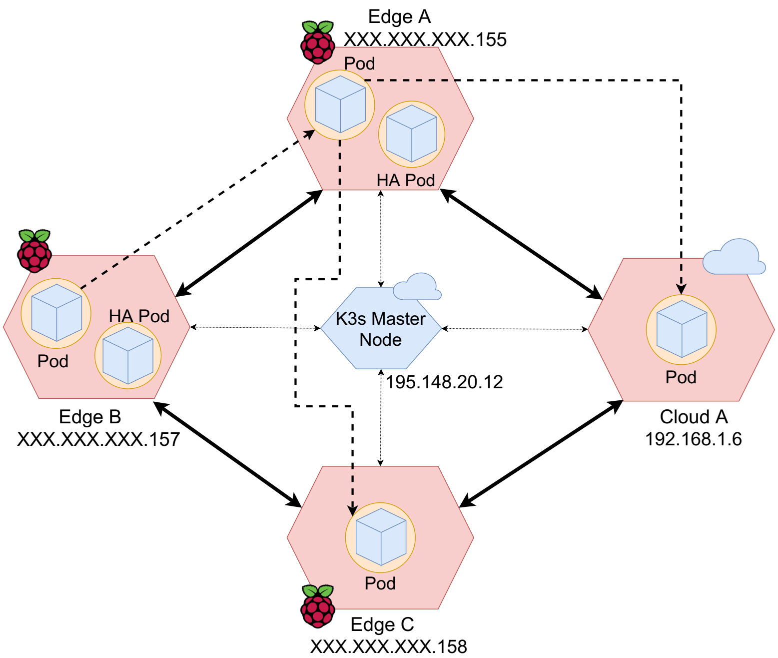


Figure 5.2: K3s pods and node deployment

cluster of edge stations (of Raspberry Pi's) and remote cloud nodes provided by CSC.

On the deployment aspect, all the elastic platform services were deployed using k3s[4]. K3s is a highly-available certified version of Kubernetes for container orchestration of production workload in resource-constrained edge subsystems. It integrates especially well for the 64-bit ARM architecture, making it suitable for deployment over Raspberry PI.

The deployment of the platform services on any Kubernetes orchestration platform is done as container "pods". We adopted the custom deployment strategy to ensure allocation of pods to the location of our choice. For example, the k3s testbed assigned cloud video inference model to a cloud node that has GPU sup-

[4]https://k3s.io

(a) input image (b) high-accuracy cloud (c) low-accuracy edge

Figure 5.3: Contrasting the inference accuracy from the standard COCO dataset [39] (Cloud and edge resources were deployed over distributed k3s nodes)

port. The node-pod assignment strategy adopted by the testbed is described in Appendix B. Figure 5.1 presents the command output and types of pod deployed and Figure 5.2 presents a high-level deployment view of pod name, node names and IP addresses of different platform services running on k3s infrastructure service. Nodes `edge-1` and `edge-2` host two pods each whereas, `edge-3` and `cloud-1` hosts only a single pod.

5.2.2 Elasticity of testbed

As per our elasticity model defined in Section 4.2, the testbed and ZETA focuses on resource elasticity. Kubernetes and by extension k3s supports variety of techniques for providing resource elasticity. More popular techniques for PaaS and IaaS auto-scaling solutions include Horizontal Pod Autoscaler (HPA)[5], Vertical Pod Autoscaler (VPA) and Libra [7]. Recently, FaaS resource elasticity can also be integrated into the Kubernetes autoscaler through Knative framework [34]. Among these, HPA is easy to use through manifests, integrated quite well into

[5] https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/

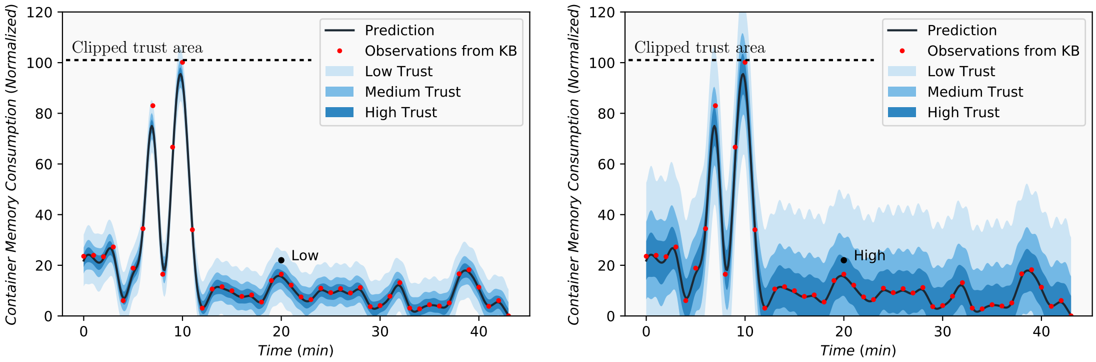
Listing 5.1: Sample (incomplete) HPA autoscaler for main-web-server

```yaml
---
apiVersion: autoscaling/v2beta2
kind: HorizontalPodAutoscaler
spec:
  scaleTargetRef:
    kind: Deployment
    name: main-web-server
  minReplicas: 1
  maxReplicas: 3
  metrics:
    - type: Resource
      resource:
        name: memory
---
```

the Kubernetes ecosystem, and most popular autoscaling solution used in the industry. An excerpt of this HPA autoscaler can be seen in Listing 5.1. As per the HPA rule, addition of new instance was done upon variation of memory allocated to a pod. High-accuracy, high-latency inferences are performed on cloud whereas low-accuracy, low-latency inference is done on edge server. Figure 5.3 presents the result of high-accuracy vs low-accuracy inference on the Microsoft COCO dataset [39]. In the baseline scenario, the decision to offload is taken by the edge-preprocessor (refer to the activity diagram in Figure 3.5(b)).
Testing criteria: The primary criteria was to use ZETA (and it's contextual trust computation and policies) to provide ZTA in edge-cloud inference in offloading. The Authentication endpoint of ZETA was called by `<<edge-node B>>` (refer to the activity diagram in Figure 3.5(b)). The other testing criteria was to examine how the trust-level output would behave when the trust configurations were varied in run-time.

5.2.3 Evaluation: ZETA with ML video inference

The preprocessor service (of the ML video inference scenario) attempted to generate an elasticity token through ZETA. If this request was accepted (i.e. ZETA generated the token), it would imply accepting the elasticity request to offload to the cloud node. The Rego policy allowed elasticity token generation on `medium` or `high` trust level output. We examined how the trust-level output (and the token generation) would behave when a) trust-sensitivity parameter was varied and b) noise parameter was varied. Varying trust sensitivity ensures there is no change



(a) Low trust level for *trust-sensitivity* = 3 (b) High trust level for *trust-sensitivity* = 9

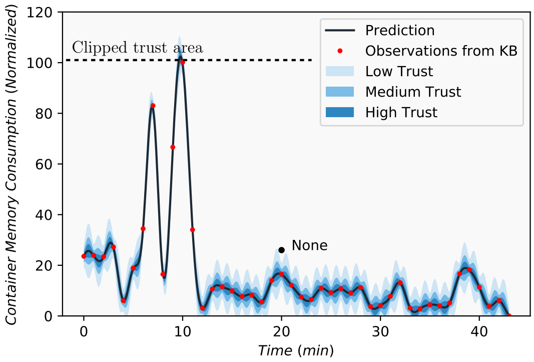
Figure 5.4: ML video inference scenario: Trust level variation by changing the *trust-sensitivity* parameter

in regression curve where as the noise parameter changes the regression output as well as the trust level. So, examining the behaviour of ZETA under these two situations is ideal.

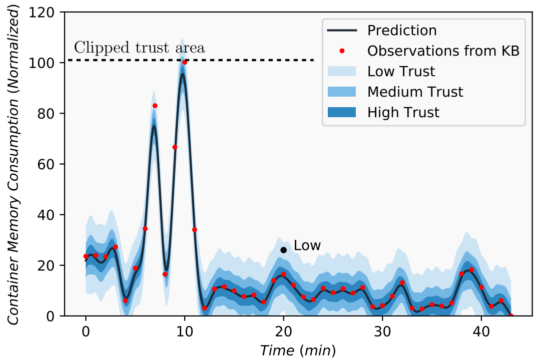
As described in Section 4.5.2, in all these scenarios, the input values were normalized and provided as a tuple. The normalized request input for time was sampled randomly which was 20. At time value=20, 22 was the normalized container memory requested (unnormalized container memory request was approx 600MB). The *trust-sensitivity* parameter, which is inversely proportional to trust levels was varied to be three and nine. These two values are selected to ensure a healthy variation in trust levels. The noise was varied for all the three possible values which are `high`, `medium` and `low`. We plot a graph of all the outputs for better visualization.

5.2.3.1 Trust-sensitivity parameter variation results

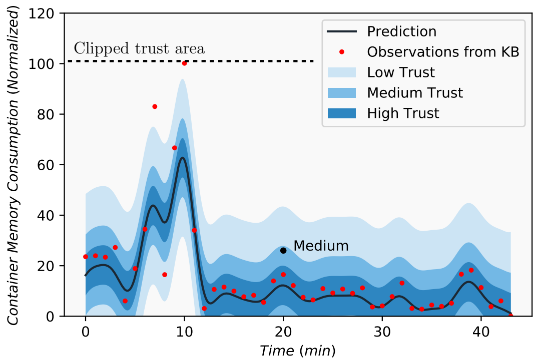
The results are presented in Figure 5.4. It shows variation of trust levels boundaries without any variation of the regression output by modifying the **trust-sensitivity** parameter. In Figure 5.4(a), the trust levels are small. The evaluated trust for (20,22) was therefore `low` and the elasticity token was not generated. In contrast, in Figure 5.4(b), the trust level are large and therefore more lenient. The evaluated trust for the same input values was `high` and ZETA permitted to offload inference request.



(a) None trust level for *noise* = 'low'



(b) Low trust level for *noise* = 'medium'



(c) Medium trust level for *noise* = 'high'

Figure 5.5: ML video inference scenario: Trust level variation by changing the `noise` parameter

5.2.3.2 Noise parameter variation results

The results are presented in Figure 5.5. The output clearly shows that noise parameter dictates the smoothness of the regression curve and the uncertainties around it. When the noise parameter is low (see Figure 5.5(a)), the trust value gets evaluated to `none` and hence the elasticity request is rejected. Whereas when the noise is 'medium' (in Figure 5.5(b)), the same input value gets evaluated to `low` and elasticity request is rejected. In Figure 5.5(c), with 'high' noise parameter, the trust is evaluated as `medium` and the elasticity request is granted.

5.3 Scenario II: GPON Monitoring System

The GPON monitoring scenario (discussed in Section 3.2) is one of the more traditional elastic platforms. Our evaluation on such a scenario demonstrates

ZETA's utilization on pre-existing and 'traditional' use-cases.

5.3.1 Testbed

The GPON testbed is implemented at `IoTCloudSamples` (the discussion and URL can be found in Section 3.2) and uses dockerized platform services. Each of these platform service has to be provisioned manually at suitable location. This is in contrast with the ML video inference scenario where the k3s provides a single-click distributed service deployment facility.

The infrastructure is kept similar to ensure that the requests and trust output results are comparable. However, since the GPON scenario is designed to be deployed at x86 machines, we utilize two virtualized edge systems (with same hardware profile) and not three Raspberry Pis. The cloud worker node is reused from Table 5.1.

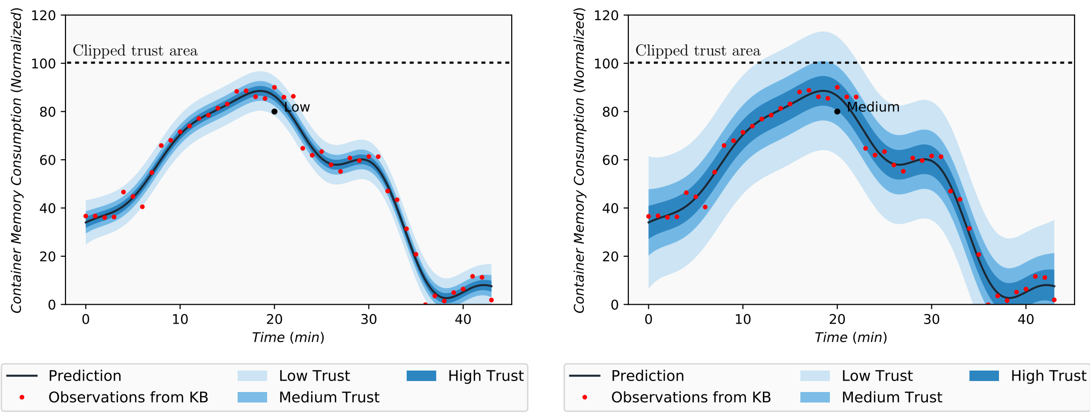
5.3.2 Elasticity of testbed

Unlike the ML video inference scenario that has HPA and VPA ingrained into Kubernetes (and by extension k3s), the GPON scenario requires external scripts to enforce elasticity. In this scenario, a script provisions a new edge message broker upon request. This worker instance of edge broker connects to main edge broker and adds horizontal scaling capability. This external script interacts with the ZETA.

Testing criteria: Similar to previous scenario, the primary criteria was to use ZETA (and it's contextual trust computation and policies) to provide ZTA. In this scenario, the Authentication endpoint of ZETA was called by external script that controls the number of edge message brokers. If the request was accepted, the script would add one extra message broker. As with the previous case, the other testing criteria was to examine how the trust-level output would behave when the trust configurations were varied.

5.3.3 Evaluation: ZETA with GPON

The external message broker provisioning script requested an elasticity token through ZETA before adding a worker message broker node. If this request was accepted (i.e. ZETA generated the token), it would imply accepting the elasticity request. Similar to the previous scenario, the Rego policy allowed elasticity token generation on `medium` or `high` trust level output. As with the previous scenario, we examined how the trust-level output (and the token generation) would behave when a) trust-sensitivity parameter was varied and b) noise parameter was varied.



(a) Low trust level for *trust-sensitivity* = 3 (b) High trust level for *trust-sensitivity* = 9

Figure 5.6: GPON scenario: Trust level variation by changing the *trust-sensitivity* parameter

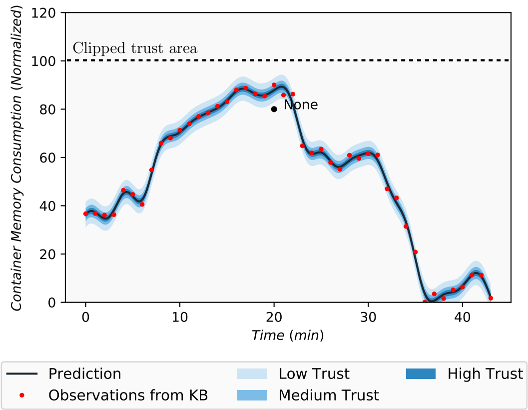
Similar to Section 4.5.2, in all these scenarios, the input values were normalized and provided as a tuple. The normalized sampling input request was (20,81). Here, 20 is the normalized time value (which is the same from the last time) and 81 is the container memory (the unnormalized edge message broker memory request was 250MB). As with the last scenario, *trust-sensitivity* parameter was varied to be three and nine and the noise was varied for all the three possible values which are `high`, `medium` and `low`.

5.3.3.1 Trust-sensitivity parameter variation results

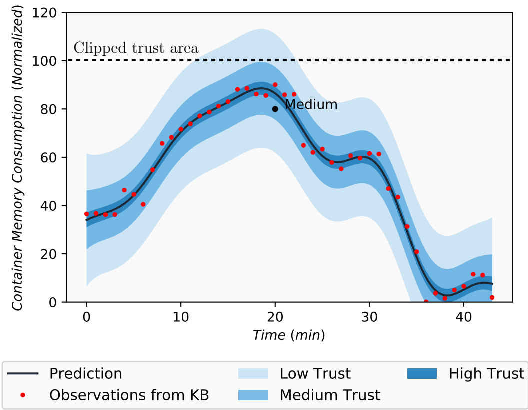
Figure 5.6 presents the graph of the results. At sensitivity 3 (in Figure 5.6(a)) the trust level output was `low` and hence, the request was rejected. In case of trust sensitivity value as 9 (in Figure 5.6(b)), the trust level output was `medium` with wider trust level boundaries and the request was accepted.

5.3.3.2 Noise parameter variation results

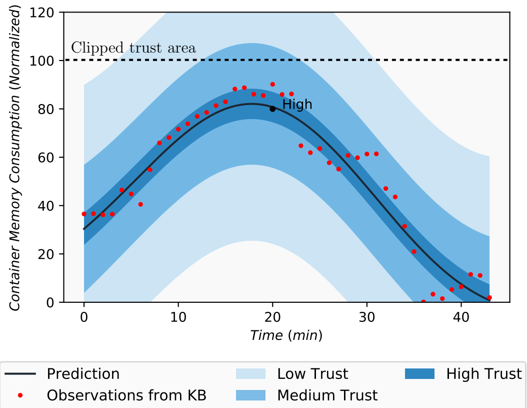
Figure 5.7(a) presents the output at `low` noise. The regression curve is tight with very low uncertainty and therefore trust levels. The output trust level evaluated was `none` and the elasticity request was rejected. Figure 5.7(b) is the output at noise value `medium`. The trust level boundaries are considerably more liberal and the trust output was evaluated as `medium` and the request was accepted. Figure 5.7(c) shows the regression curve at noise value `high`. The trust level output for the given input is `high` and therefore, the elasticity request was accepted.



(a) None trust level for *noise* = 'low'



(b) Low trust level for *noise* = 'medium'



(c) Medium trust level for *noise* = 'high'

Figure 5.7: GPON scenario: Trust level variation by changing the `noise` parameter

5.4 Performance evaluation

Since ZETA is platform agnostic, the performance of the framework should remain same in all scenarios. Therefore, in the performance evaluation, we evaluate the framework on performance metrics including CPU, end-to-end latency and memory consumption. Evaluation on each of these parameters provides a validation on real-world resource usage. For example, upon measuring end-to-end latency on the ML video inference scenario, a CSP can estimate realistic SLA guarantees while integrating ZETA framework into their products. The evaluation was performed on the same deployment infrastructure as described in Section 5.1.2.

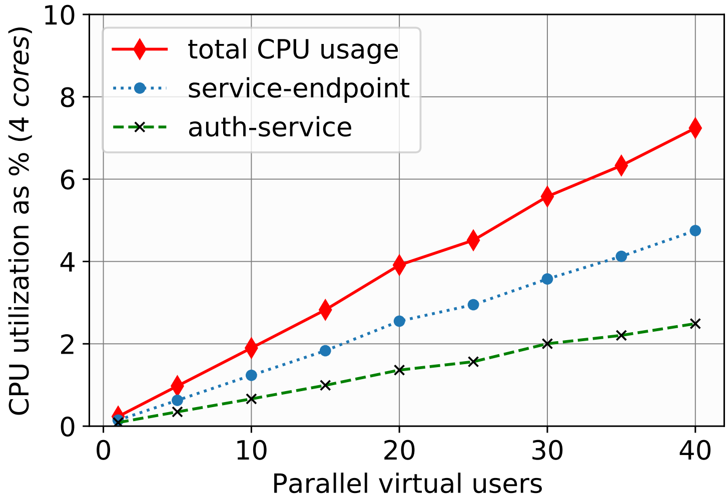


Figure 5.8: Impact on system CPU utilization for generating auth-tokens under concurrent requests

We used cAdvisor[6] and Prometheus[7] to export the runtime framework container metrics. K6[8] stress-test framework was used for API stress test via it's virtual users that enables metric reports under concurrent requests.

5.4.1 Auth token generation

• **CPU utilization**: Auth token generation is relatively computationally inexpensive process which is reflected by the results. Figure 5.8 presents the results of impact of the CPU utilization by the framework under concurrent load. We can see that the total CPU utilization grows linearly with the increase in concurrent requests. However, even under 40 parallel virtual users, the framework utilized just under 8% of the total CPU to generate auth-tokens. The memory utilization of the framework averaged **102MB** throughout the evaluation.

• **Latency**: The other important performance metric is the total response time to generate auth-tokens. Figure 5.9 presents the impact on the response time under growing concurrent requests. At 40 concurrent requests, the framework was generating over **35** auth-tokens per seconds. However, the latency of the request remained quite stable. We can see that the average response time increased from 79ms to 91ms. However, this increase remained quite small. The QoS of the service

[6] https://github.com/google/cadvisor
[7] https://prometheus.io/
[8] https://k6.io/

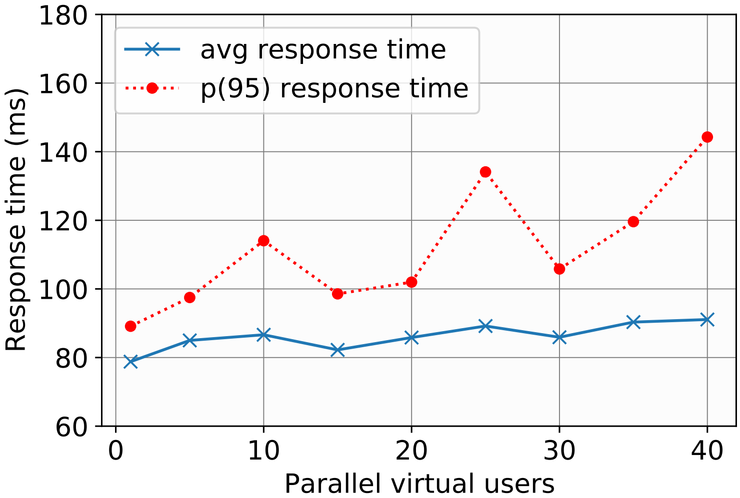


Figure 5.9: Impact on average response time for generating auth-tokens under concurrent requests

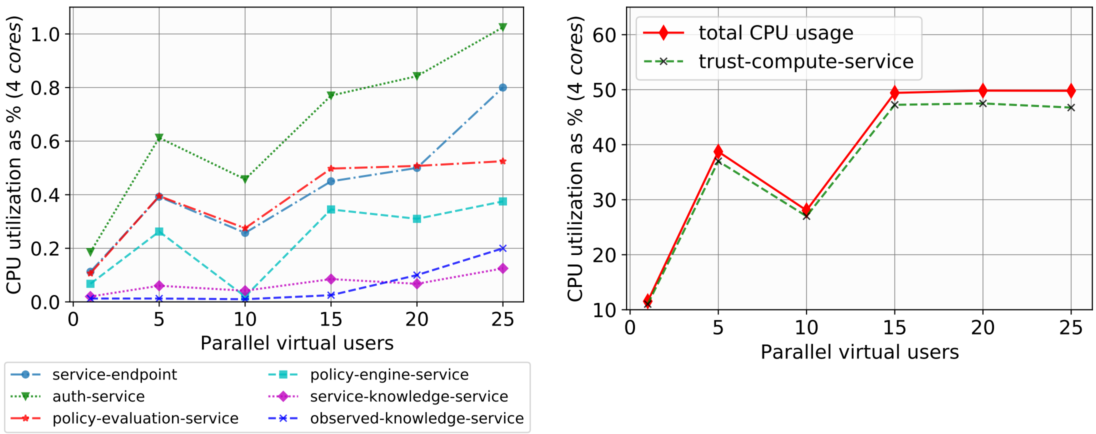
remains stable which can be inferred from the `p(95)` (95^{th} percentile) response time curve. The `p(95)` went up marginally from 89ms to 144ms even under heavy load and most requests were processed under 150ms.

5.4.2 Elasticity token generation

Elasticity token generation is significantly more resource-intensive than auth-token and requires communication between multiple components of the framework.

- **CPU utilization**: Figure 5.10 presents the impact on the CPU utilization of the system while generating the elasticity token. As more concurrent requests are made, the cumulative CPU utilization goes up significantly. As evident from Figure 5.10(a) and Figure 5.10(b), the overall CPU utilization of the framework is largely driven by `trust-compute-service`. It comprise a major chunk of the total CPU utilization and quickly becomes a bottleneck of the request/response throughput. As the number of concurrent requests increase, all the ZETA framework services show an increase in total CPU utilization. However all except one of them remains under 1% utilization. The `trust-compute-service` goes utilizing from 11% (at 1 worker) to almost 45% (at 25 concurrent workers) of the total testbed CPU. The overall CPU utilization remains hovers around 50% during peak load and remains at 11% under single request load.

- **Latency**: The latency and response rate results are plotted in Figure 5.11. Unlike the auth-token generation latency, the elasticity token generation scales lin-



(a) low CPU usage services (b) high CPU usage services

Figure 5.10: Impact on system CPU utilization for generating elasticity tokens under concurrent requests

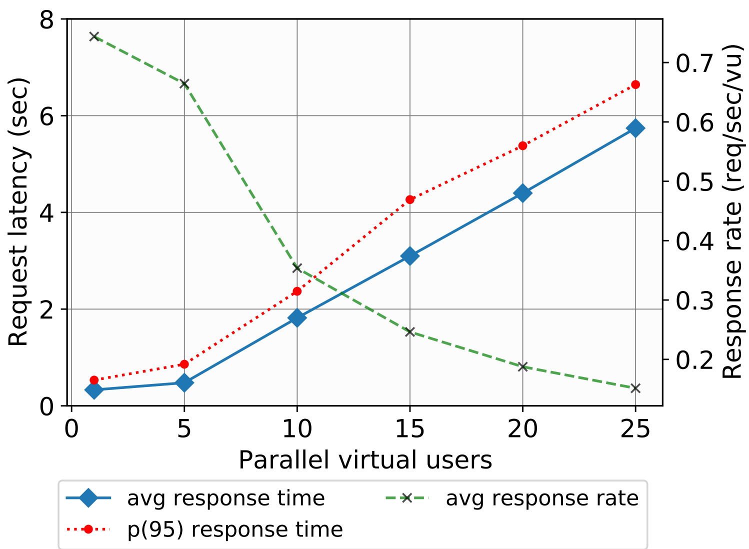


Figure 5.11: Impact on average response time for generating elasticity tokens under concurrent requests

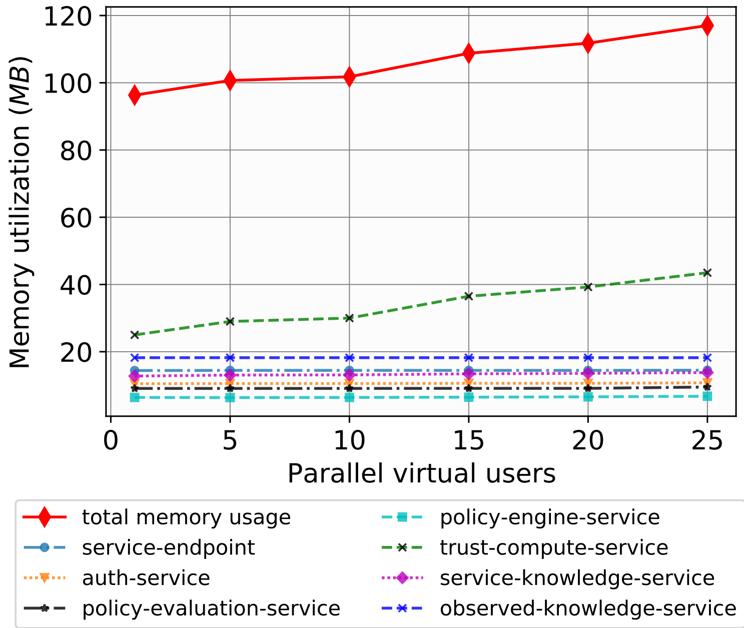


Figure 5.12: Impact on average response time for generating elasticity tokens under concurrent requests

early with concurrent requests. At a single worker, the total end-to-end elasticity token generation latency was 330ms. The latency remains almost constant at 5 concurrent workers at 448ms. The response latency showed linear increase and stood at an average of 5700ms per elasticity token at 25 concurrent workers. The `p(95)` response time curve also displayed a linear increase indicating stable performance under heavy load. The `p(95)` went up from 541ms to 6643ms. Finally, the average response rate decreased from processing 0.74 requests per second per worker to 0.15 requests per second per worker.

- **Memory Consumption**: The total and individual service's memory consumption has been plotted in Figure 5.12. Unlike CPU utilization, the memory consumption is equally distributed among the different components of the ZETA framework. The `trust-compute-service` shows slight increase in memory consumption with the increase in concurrent requests, whereas the memory consumption of remaining services remains almost constant even under increased load. As a result, the total memory consumption goes up from 96MB to 117MB during the evaluation.

5.5 Discussion

In this section, we analyse the results, discuss the implications and further provide the limitations of the framework.

5.5.1 Result comparison between scenarios

The results presented in Sections 5.2.3 and 5.3.3 provide an insight into ZETA's trust evaluation component under different scenario. It is evident that ZETA provides a contextual trust evaluation based on observed and service knowledge. In case of GPON scenario, the trust level boundaries were more relaxed compared to ML video inference scenario. This is because the uncertainties from the GPR regression were higher. It is apparent when we compare graphs in Figure 5.5(c) and Figure 5.7(c).

Another interesting observation is how outliers are treated by the trust computation algorithm. In case of ML video inference, the deviation in the memory consumption is very high. Therefore, the regression is underfitted (especially under high noise situations) in this scenario. It indicates that the default GPR trust algorithm can be substituted (extended) by the provider for a more robust algorithm. On the other hand, the deviation in memory consumption in the GPON scenario is low as it handles continuous data streams as opposed to video frames. The default GPR trust algorithm produces decent curve fitting even under high noise.

From Table 5.2, we can see ZETA's feature of contextual trust evaluation changes with both the scenarios (context) and configuration parameters. The output by varying the same parameter differs in each of the scenarios. For example, in case of ML video inference, at noise 'low', the output was rejected whereas it was accepted in the case of GPON. This is because the uncertainties in GPON scenario is higher. Moreover, it can be seen from these results that ZETA allows the providers to easily calibrate the trust parameters. Therefore, the providers can tune the trust computation algorithm depending on their use case within the scenario. For example, if the ML video inference scenario is used for airport surveillance, the provider could set the noise as high and trust sensitivity as 9 to ensure

Scenario	Trust-sensitivity = 3	Trust-sensitivity = 9	Noise = low	Noise = medium	Noise = high
MLVI	Low (✗)	High (✓)	None (✗)	Low (✗)	Medium (✓)
GPON	Low (✗)	Medium (✓)	None (✗)	Medium (✓)	High (✓)

Table 5.2: Comparison of trust-level evaluation results and ZETA's response
(✓) = elasticity request accepted / (✗) = elasticity request denied

each request gets a high-accuracy offloading. Whereas, for generic use-cases (such as a grocery store), the provider can set the noise to medium and sensitivity to a lower values (such as 3).

The evaluation also demonstrated how each parameter independently influences the trust level. The trust level bounds are inversely proportional to *accuracy* parameter. The *trust-sensitivity* parameter only affects the trust level boundaries (GPR uncertainties) but not the regression curve. This implies that it can be used for relaxing the trust level by providers in cases where the observed knowledge is harmonic or periodic. On the other hand, the *noise* parameter affects the GPR regression curve as well as the uncertainties. So providers can use this configuration parameter for managing the elasticity of highly dynamic platform microservices. Finally, the *accuracy* parameter can be used to improve performance. Therefore, each of these configurations address a different aspect of trust fine-tuning.

In the ZETA framework, the trust level output is a function of the trust algorithm. As this trust algorithm is extensible, the trust levels cannot be fixed by the framework and need to be determined by the providers of the ZETA framework. We discuss the limitation of this approach in upcoming subsection.

5.5.2 Analysis of performance evaluation results

The performance evaluation results demonstrate the stability of ZETA framework under request load. From plots in Figure 5.12 and Figure 5.10, we can confirm that the framework is compute intensive rather than memory intensive. Hence, the stakeholder must ensure framework deployment on compute intensive workloads [36]. However due to small memory footprint and low computational requirements of most of the components ZETA is an ideal candidate for edge-cloud computation offloading [87]. As all the components are distributed, the Trust computation component can be offloaded to cloud. Finally, the analysis is done on the default GPR trust algorithm whose performance can be boosted by deploying the trust component on GPU based machines. Such a deployment strategy is more optimized if the trust computation interface is extended and complex DNN algorithms are used by the stakeholder.

5.5.3 Analysis of the framework security

Another aspect is of the security of framework and potential attack surface areas. Our threat model (in Section 4.3) describes ZETA as secure and the infrastructure/platform services as insecure. The most common attack surface area in such attack models is deployment infrastructure. For example, an Arbitrary Code Execution (ACE) attack (such as CVE-2019-5736[9]) on the docker containers of the framework components can give root access of the container to the attacker. Therefore, care must be taken by the stakeholder while deploying the framework. Use of TEE backed remote container attestation techniques [20] can greatly mitigate common attacks including side-channel and cross-channel attacks. Moreover, recent researches such as [59, 80] demonstrate the feasibility and performance of trusted ML inference. Other best practises related to security of framework include using secure password for the user-endpoint configuration, periodic external certificate revocation.

We present the security benefits of the interaction model:

- In case an infrastructure service (that deploys platform service B) that already has the auth token of another platform service A is compromised, our framework will not issue any elasticity token request as it will look at the metrics of service A to compute trust.

- As the auth token is valid only for specific capabilities of deployment A, even if the trust model on the framework fails, the elasticity token will only be granted with limited capabilities.

- Developers can configure the policies centrally rather than wandering around with individual configuration files for each infrastructure deployment.

There are some other advantages of this token based approach such as the asynchronous on-demand elasticity request handling while still maintaining high security guarantees. Through ZETA framework, any service can choose to completely or partially delegate its elasticity capabilities to another service that the provider trusts.

The current design of ZETA prevents popular trust evaluation attacks include bad-mouthing attacks, sybil whitewashing attacks, and collusion attacks [81] while maintaining the subjectivity and context awareness of the trust. However, an amplified on-off attack [16] ZETA may affect trust computation algorithm.

5.5.4 Comparison with related frameworks

Table 5.3 presents a comparison of ZETA with other related frameworks. We explained these researches and the frameworks in Section 2.3. While these frameworks are not completely similar to ZETA in purpose, they support establishing trust on edge and cloud subsystems, and hence are comparable. From the table, we can seen ZETA easily is the most feature rich among them.

[9] https://www.cvedetails.com/cve/CVE-2019-5736/

Features	Papadkis et al. [55]	SegFog [22]	Davy et al. [57]	ACAS [48]	Commitment [3]	**ZETA**
Interaction	Basic REST API	Module	OAuth tokens	Module	locally installed	JWT tokens
Target resource	provider SLA	secure edge-cloud deployments	service access-control	anomaly-based predictive auto-scaling	fog-node trustworthiness	platform service elasticity
Trust Computation	Fuzzy Analytic Hierarchical Process (FAHP)	Probabilistic transitive closure	*NA*	Unsupervised isolation trees	custom	multi-dimensional GPR based
Extensibility	✗	✓	*NA*	✗	✗	✓
Trust customizability	✗	*NA*	✗	✓	✗	✓
Policy support	✓	✓	✓	✓	✗	✓
Delegation	✗	✗	✓	✗	✓	✓
Security	✗	*NA*	✓	*NA*	✓	✓
High Availability	✗	*NA*	✗	*NA*	✓	✗

Table 5.3: Comparative analysis of ZETA's features with related frameworks

5.5.5 Lessons learned

While the framework supports extensibility of trust algorithm, one of the limitations of the ZETA framework is that it requires "interpretation" of the trust levels provided by the ZETA providers. That is, any stakeholder that uses ZETA would require domain expertise in understanding the trust-level offered by the provider. As the trust level is tunable by the provider, changing the ZETA provider would require fresh interpretation and possible policy update by the users. And while the users can apply custom policies on the trust-evaluation result, this process still requires some context based knowledge by the users to determine suitable trust levels. For example, a database deployed on a VM by provider A could require `medium` trust level, whereas another database deployed in Kubernetes by provider B can require `high` trust to scale up. The framework provides no recommendation to the stakeholder about the "best" choice of trust levels.

ZETA framework is compute-intensive, which unlike memory-intensive workload is more difficult to scale-up. The main bottleneck as seen from the load test results is Trust computation component which in current iteration does not support GPU integration. Addition of this feature into ZETA would boost the trust computation performance significantly. Finally, another factor to consider is availability of the framework. There are multiple single-point-of-failure components in the framework. However, the availability can be boosted sufficiently by deploying multiple independent but geo-partitioned instances of ZETA.

ZETA utilizes HTTP header authentication which is not as secure as trusted third-party certificates. As ZETA lacks certificate based authentication, it does not integrate out-of-box with sidecar proxies such as Istio[10]. The decision to support only HTTP header ensures easier integration with computationally challenged platform services running on edge subsystem. In future, support for certificate based cloud infrastructure service authentication can be added. Through the "extern PSK" mode [27] with TLS1.3, edge infrastructure services can also use certificate based authentication and hence, HTTP header authentication can be completely phased out.

5.6 Summary

We performed and discussed the evaluation of ZETA framework in this chapter. This evaluation was performed on two real-world scenarios consisting of a cluster of edge and cloud subsystems. The encouraging results demonstrate that ZETA is successful in allowing stakeholders (both providers and users) to fine-tune the desired trust level through configuration files. ZETA also performs well under load test and was able to service very high number of requests even on a basic VM.

We also discussed limitations and the lessons learned during the evaluation of the framework. It requires fixing of trust-level computation output by the providers and is computationally expensive to operate. Moreover, few approaches like certificate based authentication can be integrated into ZETA to achieve higher security during interaction. Nonetheless, the evaluation on two real-world scenarios is extremely encouraging and demonstrates the feasibility in incorporating ZTA into the elasticity operations through ZETA framework.

[10]https://istio.io/

Chapter 6

Conclusion and future work

6.1 Conclusion

In this thesis we aim to increase the trust of elasticity operation in edge-cloud platform services by introducing ZTA paradigm into these operations. To support ZTA, we first gather functional and non-functional requirements via two real-world scenario. These two scenarios are diverse in platform and infrastructure services to ensure wide support of the extract requirements. Based on these requirements, we design the ZETA framework and the interaction model. Our platform-agnostic interaction model supports infrastructure services over common control or data plane and uses a JWT token based approach. The design of token ensures integrity during interaction and prevention from network attacks such as replay or forgery. Moreover, through the use of two different tokens, ZETA also supports delegation of elasticity between edge-cloud services.

One of the cornerstones of the ZETA framework is the context based trust computation from the derived knowledge base. ZETA builds its knowledge from the service information as well as the observed service metrics. Through the use of service and observed knowledge, the multi-dimensional trust computation determines the trust level of elasticity request. The other important aspect of ZETA is to provide trust customization to different stakeholders. While the default trust computation algorithm uses GPR to perform a multi-dimensional regression and sample trust level from it, ZETA support extensibility by allowing the provider to use custom trust algorithm. Moreover, through configuration files, the default trust computation algorithm can be fine-tuned. On the other hand, users can upload custom Rego policies to ZETA and enforce the required trust levels of their platform services at a granular level.

We performed both the feasibility and performance evaluation of the framework. The feasibility evaluation was based on two diverse and real-world scenarios. The evaluation aimed at the validation of ZETA's ability of contextual trust evaluation and variation of trust level boundaries through configuration files. The performance evaluation studied the resource consumption and bottlenecks of ZETA under sustained parallel elasticity requests. It demonstrated exceptional token generation response time even when deployed on modest VM configuration. The evaluation results and subsequent discussion indicated how ZETA can be effectively used to increase trust in the elasticity operations of edge-cloud services.

6.2 Future work

As discussed in Section 5.5.5, ZETA currently does not support GPU based inference for trust computation. Addition of dedicated GPU for faster inference along with Gaussian process frameworks such as GPflow [45] will boost the inference speed. Furthermore, use of TLS certificates with extern PSK mode can provide improved security while maintaining a low certificate verification latency [27].

Use of microservice design principles such as circuit breaker [49] and service discovery [74] can be used to improve resiliency of ZETA's components [46]. Currently, ZETA supports only one default GPR algorithm. In future, we can add multiple trust computation algorithms and present a catalogue to the stakeholder. Popular trust computation techniques such as semi-supervised trust [31], CAST [88] and particle swarm optimization [44] can be added as a configuration.

Bibliography

[1] . Iso/iec/ieee international standard - systems and software engineering – life cycle processes – requirements engineering. *ISO/IEC/IEEE 29148:2018(E)* (2018), 1–104.

[2] Abdallah, E. G., Zulkernine, M., Gu, Y. X., and Liem, C. TRUST-CAP: A trust model for cloud-based applications. In *2017 IEEE 41st Annual Computer Software and Applications Conference (COMPSAC)* (2017), vol. 2, pp. 584–589.

[3] Al-khafajiy, M., Baker, T., Asim, M., Guo, Z., Ranjan, R., Longo, A., Puthal, D., and Taylor, M. Comitment: A fog computing trust management approach. *Journal of Parallel and Distributed Computing 137* (2020), 1–16.

[4] Awad, M., and Khanna, R. *Support Vector Regression.* Apress, Berkeley, CA, 2015, pp. 67–80.

[5] Balalaie, A., Heydarnoori, A., and Jamshidi, P. Microservices architecture enables DevOps: Migration to a cloud-native architecture. *IEEE Software 33*, 3 (2016), 42–52.

[6] Balla, D., Simon, C., and Maliosz, M. Adaptive scaling of kubernetes pods. In *NOMS 2020-2020 IEEE/IFIP Network Operations and Management Symposium* (2020), IEEE, pp. 1–5.

[7] Balla, D., Simon, C., and Maliosz, M. Adaptive scaling of kubernetes pods. In *NOMS 2020 - 2020 IEEE/IFIP Network Operations and Management Symposium* (2020), IEEE Press, pp. 1–5.

[8] Berkane, M. L., Boufaida, M., and Bouzerzour, N. E. H. Modelling elastic scaling of cloud with energy-efficiency: Application to smart-university. *Journal of King Saud University - Computer and Information Sciences* (2020).

[9] Blaze, M., Feigenbaum, J., Ioannidis, J., and Keromytis, A. D. The role of trust management in distributed systems security. In *Secure Internet Programming*. Springer, 1999, pp. 185–210.

[10] Bračevac, O., Erdweg, S., Salvaneschi, G., and Mezini, M. CPL: A core language for cloud computing. In *Proceedings of the 15th International Conference on Modularity* (New York, NY, USA, 2016), MODULARITY 2016, Association for Computing Machinery, pp. 94–105.

[11] Buck, C., Olenberger, C., Schweizer, A., Völter, F., and Eymann, T. Never trust, always verify: A multivocal literature review on current knowledge and research gaps of zero-trust. *Computers & Security* (2021), 102436.

[12] Canel, C., Kim, T., Zhou, G., Li, C., Lim, H., Andersen, D. G., Kaminsky, M., and Dulloor, S. R. Scaling video analytics on constrained edge nodes, 2019.

[13] Cappos, J., Kuppusamy, T. K., Lock, J., Moore, M., and Puhringer, L. The update framework specification, May 2021.

[14] Casalicchio, E., and Perciballi, V. Auto-scaling of containers: The impact of relative and absolute metrics. In *2017 IEEE 2nd International Workshops on Foundations and Applications of Self* Systems (FAS*W)* (2017), pp. 207–214.

[15] Caturano, F., Perrone, G., and Romano, S. P. Capturing flags in a dynamically deployed microservices-based heterogeneous environment. In *2020 Principles, Systems and Applications of IP Telecommunications (IPTComm)* (2020), IEEE, pp. 1–7.

[16] Chae, Y., DiPippo, L. C., and Sun, Y. L. Trust management for defending on-off attacks. *IEEE Transactions on Parallel and Distributed Systems 26*, 4 (2015), 1178–1191.

[17] Copil, G., Moldovan, D., Truong, H.-L., and Dustdar, S. Multilevel elasticity control of cloud services. In *Service-Oriented Computing* (Berlin, Heidelberg, 2013), Springer Berlin Heidelberg, pp. 429–436.

[18] Costan, V., and Devadas, S. Intel SGX explained. *IACR Cryptol. ePrint Arch. 2016*, 86 (2016), 1–118.

[19] Darley, T., Jordan, B., and Piazza, R. Stix version 2.1. *OASIS Committee Specification Draft 01 / Public Review Draft 01* (Jul 2019).

[20] De Benedictis, M., and Lioy, A. Integrity verification of docker containers for a lightweight cloud environment. *Future Generation Computer Systems 97* (2019), 236–246.

[21] Dustdar, S., Guo, Y., Satzger, B., and Truong, H. Principles of elastic processes. *IEEE Internet Computing 15*, 5 (2011), 66–71.

[22] Forti, S., Ferrari, G.-L., and Brogi, A. Secure cloud-edge deployments, with trust. *Future Generation Computer Systems 102* (2020), 775–788.

[23] Gambetta, D. Can we trust trust? *Trust: Making and Breaking Cooperative Relations*, 13 (2000), 213–237.

[24] Garg, S., Kaur, K., Kumar, N., Kaddoum, G., Zomaya, A. Y., and Ranjan, R. A hybrid deep learning-based model for anomaly detection in cloud datacenter networks. *IEEE Transactions on Network and Service Management 16*, 3 (2019), 924–935.

[25] Gjerdrum, A. T., Johansen, H. D., Brenna, L., and Johansen, D. Diggi: A secure framework for hosting native cloud functions with minimal trust. In *2019 First IEEE International Conference on Trust, Privacy and Security in Intelligent Systems and Applications (TPS-ISA)* (2019), pp. 18–27.

[26] Gortler, J., Kehlbeck, R., and Deussen, O. A visual exploration of gaussian processes. *Distill* (2019). https://distill.pub/2019/visual-exploration-gaussian-processes.

[27] Housley, R. TLS 1.3 Extension for Certificate-Based Authentication with an External Pre-Shared Key. RFC 8773, Mar. 2020.

[28] Jain, S., Zhang, X., Zhou, Y., Ananthanarayanan, G., Jiang, J., Shu, Y., and Gonzalez, J. Rexcam: Resource-efficient, cross-camera video analytics at scale, 2019.

[29] Jangiti, S., Sriram, V. S. S., and Logesh, R. The role of cloud computing infrastructure elasticity in energy efficient management of datacenters. In *2017 IEEE International Conference on Power, Control, Signals and Instrumentation Engineering (ICPCSI)* (2017), pp. 758–763.

[30] Jarraya, Y., Eghtesadi, A., Debbabi, M., Zhang, Y., and Pourzandi, M. Cloud calculus: Security verification in elastic cloud computing platform. In *2012 International Conference on Collaboration Technologies and Systems (CTS)* (2012), pp. 447–454.

[31] Jayasinghe, U., Lee, G. M., Um, T.-W., and Shi, Q. Machine learning based trust computational model for IoT services. *IEEE Transactions on Sustainable Computing 4*, 1 (2019), 39–52.

[32] Jiang, Y., Huang, Z., and Tsang, D. H. K. Challenges and solutions in fog computing orchestration. *IEEE Network 32*, 3 (2018), 122–129.

[33] Jones, M., Arcand, B., Bergeron, B., Bestor, D., Byun, C., Milechin, L., Gadepally, V., Hubbell, M., Kepner, J., Michaleas, P., Mullen, J., Prout, A., Rosa, T., Samsi, S., Yee, C., and Reuther, A. Scalability of VM provisioning systems. In *2016 IEEE High Performance Extreme Computing Conference (HPEC)* (2016), pp. 1–5.

[34] Kaviani, N., Kalinin, D., and Maximilien, M. Towards serverless as commodity: a case of Knative. In *Proceedings of the 5th International Workshop on Serverless Computing* (2019), pp. 13–18.

[35] Khazaei, H., Ravichandiran, R., Park, B., Bannazadeh, H., Tizghadam, A., and Leon-Garcia, A. Elascale: Autoscaling and monitoring as a service. *arXiv preprint arXiv:1711.03204* (2017).

[36] Kotas, C., Naughton, T., and Imam, N. A comparison of amazon web services and microsoft azure cloud platforms for high performance computing. In *2018 IEEE International Conference on Consumer Electronics (ICCE)* (2018), pp. 1–4.

[37] Leiba, B. Oauth web authorization protocol. *IEEE Internet Computing 16*, 1 (2012), 74–77.

[38] Li, C., Tang, J., and Luo, Y. Elastic edge cloud resource management based on horizontal and vertical scaling. *The Journal of Supercomputing 76*, 10 (2020), 7707–7732.

[39] Lin, T.-Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., and Zitnick, C. L. Microsoft coco: Common objects in context. In *European conference on computer vision* (2014), Springer, pp. 740–755.

[40] Liu, F., Tong, J., Mao, J., Bohn, R., Messina, J., Badger, M., and Leaf, D. NIST cloud computing reference architecture, 2011-09-08 2011.

[41] Luna, J., Taha, A., Trapero, R., and Suri, N. Quantitative reasoning about cloud security using service level agreements. *IEEE Transactions on Cloud Computing 5*, 3 (2017), 457–471.

[42] Luna Garcia, J., Langenberg, R., and Suri, N. Benchmarking cloud security level agreements using quantitative policy trees. In *Proceedings of the 2012 ACM Workshop on Cloud Computing Security Workshop* (New York, NY, USA, 2012), CCSW '12, Association for Computing Machinery, pp. 103–112.

[43] Ma, S., Fan, C., Chuang, Y., Lee, W., Lee, S., and Hsueh, N. Using service dependency graph to analyze and test microservices. In *2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC)* (2018), vol. 02, pp. 81–86.

[44] Mao, C., Lin, R., Xu, C., and He, Q. Towards a trust prediction framework for cloud services based on pso-driven neural network. *IEEE Access 5* (2017), 2187–2199.

[45] Matthews, A. G. d. G., Van Der Wilk, M., Nickson, T., Fujii, K., Boukouvalas, A., León-Villagrá, P., Ghahramani, Z., and Hensman, J. Gpflow: A gaussian process library using tensorflow. *J. Mach. Learn. Res. 18*, 40 (2017), 1–6.

[46] Mendonca, N. C., Aderaldo, C. M., Camara, J., and Garlan, D. Model-based analysis of microservice resiliency patterns. In *2020 IEEE International Conference on Software Architecture (ICSA)* (2020), pp. 114–124.

[47] Michels, J., Hare, K., Kulkarni, K., Zuzarte, C., Liu, Z., Hammerschmidt, B., and Zemke, F. The new and improved SQL:2016 standard. *ACM SIGMOD Record 47* (2018), 51–60.

[48] Moghaddam, S. K., Buyya, R., and Ramamohanarao, K. Acas: An anomaly-based cause aware auto-scaling framework for clouds. *Journal of Parallel and Distributed Computing 126* (2019), 107–120.

[49] Montesi, F., and Weber, J. Circuit breakers, discovery, and API gateways in microservices, 2016.

[50] Moreira, D. A. B., Marques, H. P., Costa, W. L., Celestino, J., Gomes, R. L., and Nogueira, M. Anomaly detection in smart environments using AI over fog and cloud computing. In *2021 IEEE 18th Annual Consumer Communications Networking Conference (CCNC)* (2021), pp. 1–2.

[51] Moreno-Vozmediano, R., Montero, R. S., Huedo, E., and Llorente, I. M. Efficient resource provisioning for elastic cloud services based on machine learning techniques. *J. Cloud Comput. 8*, 1 (Dec. 2019).

[52] Neuman, B., and Ts'o, T. Kerberos: an authentication service for computer networks. *IEEE Communications Magazine 32*, 9 (1994), 33–38.

[53] Osborn, B., McWilliams, J., Beyer, B., and Saltonstall, M. Beyondcorp: Design to deployment at Google. *;login: 41* (2016), 28–34.

[54] Ostberg, P., Byrne, J., Casari, P., Eardley, P., Anta, A. F., Forsman, J., Kennedy, J., Le Duc, T., Marino, M. N., Loomba, R., Lopez Pena, M. Ã., Veiga, J. L., Lynn, T., Mancuso, V., Svorobej, S., Torneus, A., Wesner, S., Willis, P., and Domaschka, J. Reliable capacity provisioning for distributed cloud/edge/fog computing applications. In *2017 European Conference on Networks and Communications (EuCNC)* (2017), pp. 1–6.

[55] Papadakis-Vlachopapadopoulos, K., GonzÃ¡lez, R. S., Dimolitsas, I., Dechouniotis, D., Ferrer, A. J., and Papavassiliou, S. Collaborative SLA and reputation-based trust management in cloud federations. *Future Generation Computer Systems 100* (2019), 498–512.

[56] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research 12* (2011), 2825–2830.

[57] Preuveneers, D., and Joosen, W. Towards multi-party policy-based access control in federations of cloud and edge microservices. In *2019 IEEE European Symposium on Security and Privacy Workshops (EuroS PW)* (2019), pp. 29–38.

[58] Qolomany, B., Mohammed, I., Al-Fuqaha, A., Guizani, M., and Qadir, J. Trust-based cloud machine learning model selection for industrial IoT and smart city services. *IEEE Internet of Things Journal 8*, 4 (2021), 2943–2958.

[59] Quoc, D. L., Gregor, F., Arnautov, S., Kunkel, R., Bhatotia, P., and Fetzer, C. SecureTF: A secure tensorflow framework. In *Proceedings of the 21st International Middleware Conference* (New York, NY, USA, 2020), Middleware '20, Association for Computing Machinery, pp. 44–59.

[60] Qureshi, K. N., Jeon, G., and Piccialli, F. Anomaly detection and trust authority in artificial intelligence and cloud computing. *Computer Networks 184* (2021), 107647.

[61] Raj, R., and Truong, L. On analysis of security and elasticity dependency in IIoT platform services. In *2021 IEEE International Conference on Services Computing (SCC)* (United States, 2021), IEEE.

[62] Rani, D., and Ranjan, R. K. A comparative study of SaaS, PaaS and IaaS in cloud computing. *International Journal of Advanced Research in Computer Science and Software Engineering 4*, 6 (2014).

[63] Rasmussen, C. E., and Williams, C. K. I. *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press, 2005.

[64] Redmon, J., and Farhadi, A. Yolov3: An incremental improvement, 2018.

[65] Reschke, J. The 'Basic' HTTP Authentication Scheme. RFC 7617, Sept. 2015.

[66] Ribeiro, A. V., Sampaio, L. N., and Ziviani, A. Affinity-based user clustering for efficient edge caching in content-centric cellular networks. In *2018 IEEE Symposium on Computers and Communications (ISCC)* (2018), pp. 00474–00479.

[67] Rose, S., Borchert, O., Mitchell, S., and Connelly, S. Zero trust architecture, NIST special publication 800-207, 8 2020.

[68] Sahli, H., Belala, F., and Bouanaka, C. A BRS-based approach to model and verify cloud systems elasticity. *Procedia Computer Science 68* (2015), 29–41. 1st International Conference on Cloud Forward: From Distributed to Complete Computing.

[69] Salah, T., Jamal Zemerly, M., Yeun, C. Y., Al-Qutayri, M., and Al-Hammadi, Y. The evolution of distributed systems towards microservices architecture. In *2016 11th International Conference for Internet Technology and Secured Transactions (ICITST)* (2016), pp. 318–325.

[70] Schulz, E., Speekenbrink, M., and Krause, A. A tutorial on gaussian process regression: Modelling, exploring, and exploiting functions. *Journal of Mathematical Psychology 85* (2018), 1–16.

[71] Sethi, A. A review paper on levels, types & techniques in software testing. *International Journal of Advanced Research In Computer Science 8*, 7 (2017).

[72] Shi, W., Cao, J., Zhang, Q., Li, Y., and Xu, L. Edge computing: Vision and challenges. *IEEE Internet of Things Journal 3*, 5 (2016), 637–646.

[73] Smith, R. E. A contemporary look at saltzer and schroeder's 1975 design principles. *IEEE Security Privacy 10*, 6 (2012), 20–25.

[74] Stubbs, J., Moreira, W., and Dooley, R. Distributed systems of microservices using docker and serfnode. In *2015 7th International Workshop on Science Gateways* (2015), IEEE, pp. 34–39.

[75] Syed, H. J., Gani, A., Ahmad, R. W., Khan, M. K., and Ahmed, A. I. A. Cloud monitoring: A review, taxonomy, and open research issues. *Journal of Network and Computer Applications 98* (2017), 11–26.

[76] Tan, Y., Nguyen, H., Shen, Z., Gu, X., Venkatramani, C., and Rajan, D. PREPARE: Predictive performance anomaly prevention for virtualized cloud systems. In *2012 IEEE 32nd International Conference on Distributed Computing Systems* (2012), pp. 285–294.

[77] Truong, H.-L. Using IoTCloudSamples as a software framework for simulations of edge computing scenarios, 07 2019.

[78] Tsigkanos, C., Garriga, M., Baresi, L., and Ghezzi, C. Cloud deployment tradeoffs for the analysis of spatially distributed internet of things systems. *ACM Transactions on Internet Technology 20*, 2 (Apr. 2020).

[79] Varghese, B., and Buyya, R. Next generation cloud computing: New trends and research directions. *Future Generation Computer Systems 79* (2018), 849–861.

[80] Volos, S., Vaswani, K., and Bruno, R. Graviton: Trusted execution environments on GPUs. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)* (Carlsbad, CA, Oct. 2018), USENIX Association, pp. 681–696.

[81] Wang, J., Jing, X., Yan, Z., Fu, Y., Pedrycz, W., and Yang, L. T. A survey on trust evaluation based on machine learning. *ACM Computing Surveys 53*, 5 (Sept. 2020).

[82] Wang, N., Varghese, B., Matthaiou, M., and Nikolopoulos, D. S. ENORM: A framework for edge node resource management. *IEEE Transactions on Services Computing 13*, 6 (2020), 1086–1099.

[83] Wen, Z., Lin, T., Yang, R., Ji, S., Ranjan, R., Romanovsky, A., Lin, C., and Xu, J. Ga-par: Dependable microservice orchestration framework for geo-distributed clouds. *IEEE Transactions on Parallel and Distributed Systems 31*, 1 (2020), 129–143.

[84] Wu, H., Zhang, Z., Guan, C., Wolter, K., and Xu, M. Collaborate edge and cloud computing with distributed deep learning for smart city internet of things. *IEEE Internet of Things Journal 7*, 9 (2020), 8099–8110.

[85] Xiu, D., and Liu, Z. A formal definition for trust in distributed systems. In *Information Security* (Berlin, Heidelberg, 2005), J. Zhou, J. Lopez, R. H. Deng, and F. Bao, Eds., Springer Berlin Heidelberg, pp. 482–489.

[86] Zhang, Q., Liu, L., Pu, C., Dou, Q., Wu, L., and Zhou, W. A comparative study of containers and virtual machines in big data environment. In *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)* (2018), pp. 178–185.

[87] Zhang, X., and Debroy, S. Energy efficient task offloading for compute-intensive mobile edge applications. In *ICC 2020 - 2020 IEEE International Conference on Communications (ICC)* (2020), pp. 1–6.

[88] Zhou, P., Gu, X., Zhang, J., and Fei, M. A priori trust inference with context-aware stereotypical deep learning. *Knowledge Based Systems 88*, C (Nov. 2015), 97–106.

Appendix A

Appendix: GPON data schema and sample

```
TIME: record time
PROVINCECODE: code of the network sector
DEVICEID: id of device
IFINDEX: interface id
FRAME: frame id
SLOT: slot id
PORT: port number
ONUINDEX: onu index
ONUID: network interface
SPEEDIN:  traffic moved into an ONU(network interface)
    ↪ measured in bit/s
SPEEDOUT: traffic moved out an ONU(network interface)
    ↪ measured in bit/s
```

Listing A.1: GPON data schema

PROVINCECODE	DEVICEID	IFINDEX	FRAME	SLOT	PORT	ONUINDEX	ONUID	TIME	SPEEDIN	SPEEDOUT
HKD	2222771642618	6828878457269	1	1	13	20	222277164261810113020	01/08/2019 11:38:33	2992	2947
HKD	2222771642618	6828878457269	1	1	13	21	222277164261810113021	01/08/2019 11:38:33	31271245	1414548
HKD	2222771642618	6828878457269	1	1	13	2	222277164261810113002	01/08/2019 11:38:33	677495	20658
HKD	2222771642618	6828878457269	1	1	13	3	222277164261810113003	01/08/2019 11:38:33	0	0
HKD	2222771642618	6828878457269	1	1	13	1	222277164261810113001	01/08/2019 11:38:33	1366458	42488

Table A.1: Sample GPON data

Anonymized GPON data taken from : https://version.aalto.fi/gitlab/bigdataplatforms/cs-e4640/-/tree/master/data/onudata

Appendix B

Appendix: K3s deployment strategies

B.1 Edge deployment strategy

Scheduling eviction strategy was used to reassign pods onto the correct infrastructure[1]. The most recommended approach for this is through the use of `kubernetes.spec.template.spec.nodeSelector` k8s configuration. In this approach, we first assign the role (`node-role.kubernetes.io/worker`) and then label the nodes. Finally, the pods can the be strategically deployed using a configuration manifest similar to Listing B.1.

Listing B.1: Sample k3s nodeSelector deployment

```
spec:
replicas: 1
selector:
  matchLabels:
    app: edge-preprocessor
template:
  metadata:
    labels:
      app: edge-preprocessor
  spec:
    nodeSelector:
      deploymentType: "edgeWorker.rdsea.csc"
    containers:
      - name: edge-preprocessor
```

[1] https://kubernetes.io/docs/concepts/scheduling-eviction/assign-pod-node/

B.2 GPU over K3s

Nvidia GPU's are not natively supported by the `containerd`[2], which is the default runtime of k3s and k8s. One of the techniques to support GPU is by switching the container runtime to docker container runtime[3]. However, both k3s and k8s are now use `containerd` as their default runtime. Moreover, smaller footprint of `containerd` makes it more ideal for our testbed infrastructure over spread across cloud and Raspberry Pi's. Additionally, one of our requirements was support of both GPU and non-GPU type of infrastructure together.

Therefore, to enable GPU support[4], containerd's configurations require adding the lines from Listing B.2 *only* on the GPU nodes.

Listing B.2: Sample k3s nodeSelector deployment

```
[plugins.cri.containerd.runtimes.runc]
#--- - changed from 'io.containerd.runc.v2' for GPU support
runtime_type = "io.containerd.runtime.v1.linux"

#--- - added for GPU support
[plugins.linux]
runtime = "nvidia-container-runtime"
```

Finally, we use the scheduling eviction strategy described in Section B.1 to deploy pods strategically on GPU and non-GPU resources. A sample labelling strategy has been show in listing below:

```
$ k3s kubectl label node NODE_NAME GPU_WORKER_NODE_NAME=XXX
```

[2] https://containerd.io/
[3] https://www.docker.com/products/container-runtime
[4] https://dev.to/mweibel/add-nvidia-gpu-support-to-k3s-with-containerd-4j17